sigmadue CU-02 & MP-01

Firmware Function Blocks Library

Function Block Library
M.U. CU02-FFBL 06/11.05
Cod. J50 - 636 – 1ACU02 - E

Copyright © 2007, 2011 Ascon Tecnologic Srl

All rights reserved

No part of this document may be stored in a retrieval system, or transmitted in any form, electronic or mechanical, without prior written permission of Ascon Tecnologic Srl.

Ascon has used the best care and effort in preparing this manual and believes that the information contained in this publication is accurate.
As Ascon Tecnologic continues to improve and develop products, the information contained in this manual may also be subject to change. Ascon Tecnologic reserves the right to change such information without notice.
Ascon Tecnologic makes no warranty of any kind, expressed or implied, with regard to the documentation contained in this manual. Ascon Tecnologic shall not be liable in any event - technical and publishing error or omissions - for any incidental and consequential damages, in connection with, or arising out of the use of this manual.

sigma*due*®, *gamma**due*® *and delta**due*®, are trademarks of Ascon Tecnologic Srl.

All other trade names or product names are trademarks or registered trademarks.

Ascon Tecnologic srl

Headquarters: via Indipendenza 56, 27029 Vigevano (PV)

Milan office: Via Falzarego 9/11, 20021 Baranzate (MI)

Phone +39 02 333 371

Fax +39 02 350 4243

www.ascontecnologic.com

sales@ascontecnologic.com

INDEX

Note: The Digital Inputs and Outputs of the MP-01 Control Unit are explained in the "*MP-01 User Manual*"

Preface vii

Chapter 1

AsconCTRLFb-Embedded 1

1-1 Purpose 1
1-2 Description 1
1-3 PID implementation 1
1-4 Description of the individual Function Blocks 4
1-4-1 CTRL_PID 4
1-4-2 CTRL_MV 6
1-4-3 CTRL_HCMV 7
1-4-4 CTRL_SPLITMV 9
1-4-5 CTRL_SRV 11
1-4-6 CTRL_SRV_POS 12
1-4-7 CTRL_TPO 14
1-5 Example task 15

Chapter 2

MODBUS TCP/IP 16

2-1 Modbus TCP/IP Server agent 17
2-1-1 ENABLE_MODBUS_TCP_SERVER 20
2-1-2 CLOSE_MODBUS_TCP_SERVER 20
2-1-3 MB_TCP_CONN_STATUS 21
2-1-4 MB_TCP_GET_CONN_BY_ADDR 22
2-1-5 MB_TCP_GET_CONN_CONFIG 23
2-1-6 MB_TCP_CLOSE_CONN 24
2-1-7 Configurations using the Terminal VT100 initial setup session . 25
2-1-8 Application example of the Modbus TCP/IP Server agent . . 26
2-1-9 Global application variables 27
2-1-10 HW port initialisation 27
2-1-11 Example code 28

Chapter 3

Serial communications 30

3-1 Serial communication configuration 30
3-1-1 OPEN_SERIAL_COMM 31
3-1-2 CLOSE_SERIAL_COMM 32
3-2 Modbus Slave protocol 33
3-2-1 MODBUS_SLAVE_SETTINGS 37
3-2-2 MODBUS_SLAVE_STATUS 38
3-3 Data access function for Modbus slaves 39
3-3-1 MODBUS_GET_SLAVE_DATA 39
3-3-2 MODBUS_SET_WORD_DATA 40
3-3-3 MODBUS_SET_DWORD_DATA 41
3-3-4 MODBUS_GET_DIGITAL_SLAVE 42
3-3-5 MODBUS_SET_DIGITAL_SLAVE 43
3-4 Master modbus protocol 44
3-4-1 MODBUS_MASTER_EXECUTE 46
3-4-2 MODBUS_MASTER_STATUS 47
3-4-3 Application example of the modbus communication port . . . 48
3-4-4 Global application variables 48
3-4-5 HW port initialisation 48
3-4-6 Master code 50
3-4-7 Slave code 51
3-5 ASCII serial 53
3-5-1 CONV_ASCII_TO_CHAR 53
3-5-2 CONV_CHAR_TO_ASCII 53
3-5-3 SERIAL_IO_CONFIG 54
3-5-4 SERIAL_IO_READ 55
3-5-5 SERIAL_IO_READ_BYTE 56
3-5-6 SERIAL_IO_WRITE 57
3-5-7 SERIAL_IO_WRITE_BYTE 58

Chapter 4

CanOPEN (for CU-02 unit only) 59

4-1 Canopen FBs 59
- 4-1-1 CanOPEN_Autolearn 59
- 4-1-2 CanOPEN_Device_Compare 60
- 4-1-3 CanOPEN_Get_Device_Com_Par 61
- 4-1-4 CanOPEN_get_Device_id 62
- 4-1-5 CanOPEN_Network_Compare 62
- 4-1-6 CanOPEN_Network_Status 63
- 4-1-7 CanOPEN_Net_Control_Start 64
- 4-1-8 CanOPEN_Scan_Device_Com 65
- 4-1-9 CanOPEN_Scan_Device_Id 65
- 4-1-10 CAN_ENABLE_CYCLIC_SYNC 66
- 4-1-11 CAN_GET_CANOPEN_KERNEL_STATE 67
- 4-1-12 CAN_GET_LOCAL_NODE-ID 67
- 4-1-13 CAN_GET_STATE 68
- 4-1-14 CAN_PDO_READ8 69
- 4-1-15 CAN_PDO_WRITE8 70
- 4-1-16 CAN_RECV_BOOTUP 71
- 4-1-17 CAN_RECV_BOOTUP_DEV 72
- 4-1-18 CAN_RECV_EMCY 72
- 4-1-19 CAN_RECV_EMCY_DEV 73
- 4-1-20 CAN_REGISTER_COBID 75
- 4-1-21 CAN_SDO_READ8 75
- 4-1-22 CAN_SDO_READ_STR 76
- 4-1-23 CAN_SDO_WRITE8 77
- 4-1-24 CAN_SDO_WRITE_STR 78
- 4-1-25 CAN_SEND_SYNC 79
- 4-1-26 CAN_WRITE_EMCY 80
- 4-1-27 GET_SUPPORTED_DIAG 81
- 4-1-28 NET_ASCON_ID 82
- 4-1-29 NG_RTR 83

Chapter 5

PROFIBUS (for CU-02 unit only) 84

5-1 PROFIBUS-DP Slave Agent 84
- 5-1-1 PROFI_ENABLE 86
- 5-1-2 PROFI_STATUS 87
- 5-1-3 PROFI_SEND_TO_MASTER 87

Chapter 6

Memory Copy Functions 88

6-1 Copy Function Blocks 88
- 6-1-1 MEMCOPY_FROM_M 88
- 6-1-2 MEMCOPY_TO_M 90
- 6-1-3 MEMCPY_I_TO_M 91
- 6-1-4 MEMCPY_M_TO_M 92
- 6-1-5 MEMCPY_M_TO_Q 93
- 6-1-6 MEMCPY_Q_TO_M 94

Chapter 7

Real Time Clock 95

7-1 Description of the individual Function Blocks 95
7-1-1 RTC_GET_VALUES 95
7-1-2 RTC_SETUP 96

Chapter 8

Watchdog 97

8-1 Description of the individual Function Blocks 97
8-1-1 WATCHDOG_SET 97
8-1-2 WATCHDOG_STATUS 99

Chapter 9

Random number generators 100

9-1 Description of the individual Function Blocks 100
9-1-1 RAND 100

Chapter 10

Type conversion (FLATTEN IEEE – REAL and vice versa) . . 101

10-1 Description of the individual Function Blocks 101
10-1-1 ASCON_FLATTEN_TO_REAL 101
10-1-2 ASCON_REAL_TO_FLATTEN 102

Chapter 11

Counter Block (for MP-01 unit only) 103

11-1 RESET_PULSE_COUNTER 103

Chapter 12

TC_Calibrate (for MP-01 unit only) 104

Appendix A

Reference documents 105

Preface

The products described in this manual should be installed, operated and maintained only by qualified application programmers and software engineers who are almost familiar with EN 61131-3 concepts of PLC programming, automation safety topics and applicable national standards.

Introduction

The functions in OpenPCS made available by Ascon hardware can be managed through the Infoteam product, which enables a procedure to define Function Blocks to be directly used in the IEC61131-3 programming environment and to be executed by calling a function implemented in the firmware on the CPU card. These Function Blocks for example will enable to manage communication through the auxiliary serial ports according to Modbus protocol or to use the I/O digital ports on the cards.

Using this manual

Specifications within the text of this manual are given in the International System of Units (SI), with non SI equivalents in parentheses.

Fully Capitalized words within the text indicate markings found on the equipment.

Words in **bold** style within the text indicate markings found in the Configuration Tools.

Warnings, Cautions and Notes are used to emphasize critical instructions:

DANGER!

Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

WARNING

Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.

Caution
Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

Note: Highlights important information about an operating procedure or the equipment.

Purpose

The present document is intended to describe structure and use of the Function Blocks implemented in the CPU firmware devoted to manage all the built-in functionalities of the CU-02 and MP-01 device. Thanks to these Function Blocks, using OpenPCS IEC61131-3 programming tool, is possible to access all the peripheral, and to manage all the basic control present in the device.

Caution
The function blocks of the various libraries are valid for both CU-02 and MP-01 units except when it is expressly indicated.

Current Documentation on the Internet

Make sure you are always working with the latest version of this document.

ASCON spa reserves the right to make changes to its products in the name of technological advancement. New manual revisions, when published, and can be found online at:

http://www.ascon.it

Chapter 1
AsconCTRLFb-Embedded

1-1 Purpose

The purpose of this document is to describe the control Function Blocks available in the Ascon spa control units.

1-2 Description

The Function Blocks described below represent the totality of basic control functions made available by Ascon devices in terms of functions required in process control applications. All FBs are embedded: i.e. they do not belong to any specific IEC61131 library, but they are always present in the device firmware. The following is the list of present FBs.

CTRL_PID	PID algorithm
CTRL_MV	Automan Station for single action regulation
CTRL_HCMV	Automan Station for heat and cool regulation
CTRL_SPLITMV	Automan Station for heat and cool regulation with split range algorithm
CTRL_SRV	Servomotors algorithm
CTRL_SRV_POS	Servomotors algorithm close loop (positioner)
CTRL_TPO	Time proportional output

For further details, see the specific documentation of the individual module.

1-3 PID implementation

The implementation of the PID algorithm is based on a standard PID implementation in the latest generation Ascon instruments, with the implementation of a Setpoint_Weighting scheme based on an evolved Overshoot Control algorithm. Theoretical references on control theory are confined to describing the structure of the system.

The implemented PID algorithm is in its most complete (and complex) form and the dynamic relationship binding the **setpoint**, the process **value** and the **control variable** is the following:

$$U=K\left(\beta Y_{SP}-Y+\frac{1}{sT_i}E-\frac{sT_d}{1+\frac{sT_d}{N}}Y\right)$$

where:

U control variable

K gain

β overshoot control

$\mathbf{Y_{SP}}$ setpoint

Y value

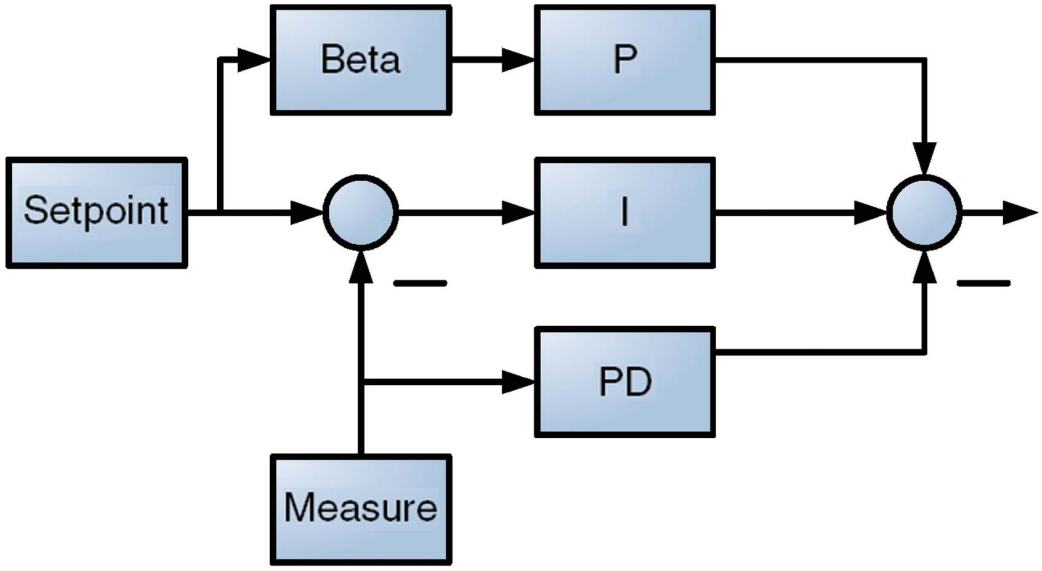
$\mathbf{T_i}$ integral time

$\mathbf{T_d}$ derivative time

E error (setpoint - value)

N filter constant for numerical computation of derivative

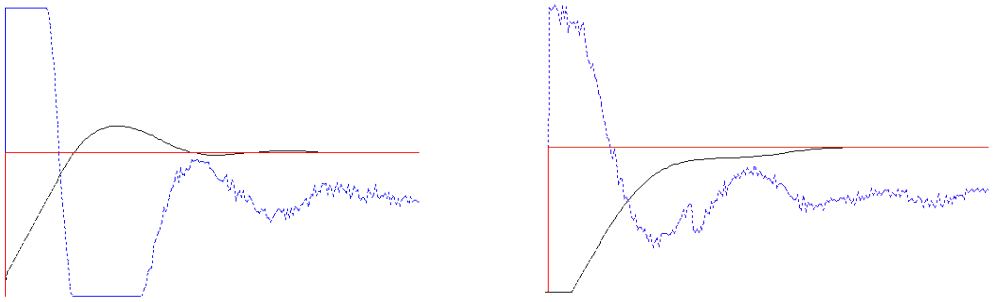
The basic structure of the controller is illustrated below:



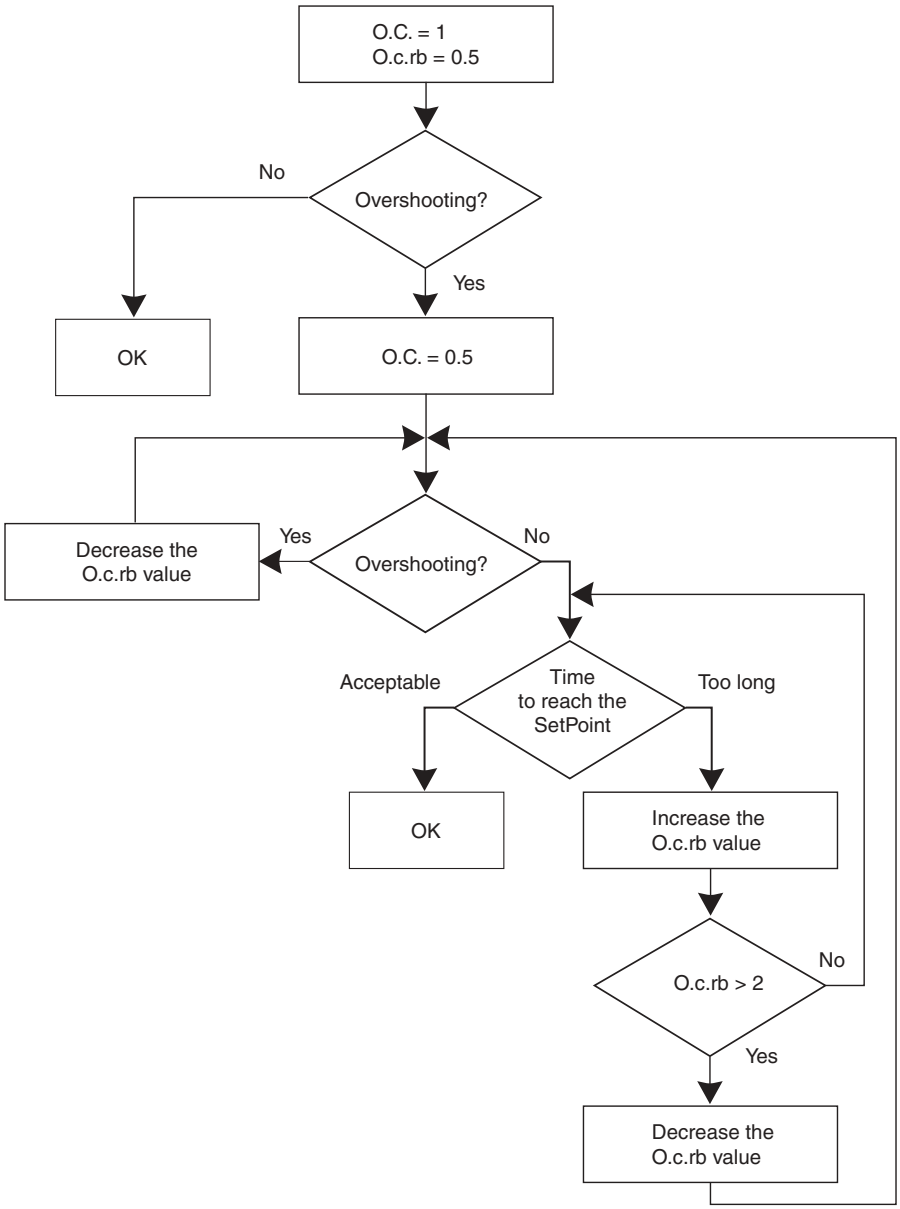
The functional block has other auxiliary inputs used to add specific features: manual reset, bumpless command, integral stop action command, etc..

The “overshoot control” algorithm implemented in the FB optimises the control of over/undershoots which may occur when the Setpoint is changed. The available parameters intended for this purpose are two: *O.C.* and *O.C.rb*. The first one represents the actual control parameter of over/undershoots, whereas the second one defines a band near the setpoint where the PID automatically excludes this control. In particular, decreasing the O.C. value means decreasing the overshoot effect and accordingly increasing the time needed to reach the Setpoint. Increasing O.C.rb means increasing the area near the Setpoint where PID returns to its natural dynamics and accordingly decreasing the time needed to reach the Setpoint.

The two figures below show the effect of the overshoot control algorithm on a process as a result of setpoint step variation. In the left figure the algorithm is excluded (O.C. = 1.0).



The following is a flow chart indicating the required procedure to obtain correct “O.C.” and “O.C.rb” parameter values, starting from their default values.

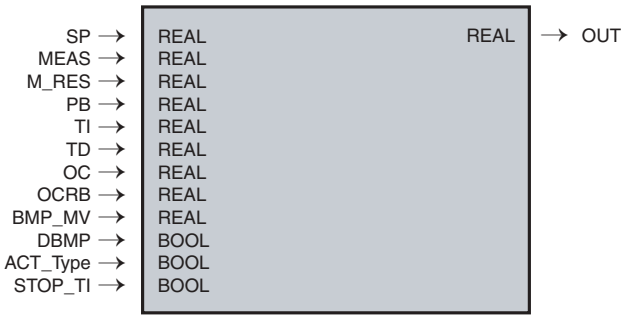


1-4 Description of the individual Function Blocks

1-4-1 CTRL_PID

FB Prototype

CTRL_PID



Input parameters

Label	Type	Description
SP	REAL	Setpoint value [0.0... 100.0] (default value = 0.0)
MEAS	REAL	Measure value [0.0... 100.0] (default value = 0.0)
M_RES	REAL	Manual Reset [0.0... 100.0] (default value = 50.0)
PB	REAL	Proportional Band [%](default value = 5.0)
TI	REAL	Integral Time [seconds] (default value = 180.0)
TD	REAL	Derivative Time [seconds] (default value = 40.0)
OC	REAL	Overshoot Control (default value = 1.0)
OCRB	REAL	Overshoot Control Relative Band (default value = 0.5)
BMP_MV	REAL	Value used for the BUMPLESS functionality (default value = 0.0)
DBMP	BOOL	BUMPLESS command input (default value = FALSE)
ACT_Type	BOOL	PID Action Type (default value = FALSE)
STOP_TI	BOOL	External command to stop the integral calculation (default value = FALSE)

Output parameters

Label	Type	Description
OUT	REAL	Controller Output [0.0…100.0] (default value = 0.0)

Description

This function block implements the PID algorithm typically used by the Ascon controllers. It needs as input the process variables (*SP* and *MEAS*) in [%] format, the parameters (PB*, TI, TD, M_RES, OC, OCRB, ACT_Type*), and the bumpless parameters (BMP_MV and DBMP). As output provides the control value *MV* in 0.0… 100.0 range.

Here is reported a detailed description of the parameters:

Parameter	Description
SP	Setpoint Value in [0.0... 100.0]
MEAS	Measure Value in [0.0... 100.0]
PB	Proportional band coefficient that multiplies the error (SP - MEAS)
M_RES	Manual reset. It is the control output value when PV = MEAS in a PD only algorithm (lack of the integral term)
TI	Integral time value. It is the time required by the integral term to generate an output equivalent to the proportional term
TD	Derivative time. It is the time required by the proportional term P to repeat the output provided by the derivative term D
OC	Overshoot control. It is the span of the action of the overshoot control. Setting lower values (1.00... 0.01) the overshoot generated by a set point change is reduced. The overshoot control does not affect the effectiveness of the PID algorithm. Setting 1 the overshoot control is disabled
OCRB	Overshoot control relative band. It defines a zone across the setpoint where the PID algorithm is not affected by the overshoot control
BMP_MV	Value used for the BUMPLESS calculation
DBMP	Digital Input indicating when must be performed the adjustment of the PID terms for the BUMPLESS functionality
ACT_Type	Selection of the control output direction: direct (True) or inverse (False)
STOP_TI	External command to stop the integral calculation
MV	Controller Output Value

1-4-2 CTRL_MV

FB Prototype

CTRL_MV

Input parameters

Label	Type	Description
MV	REAL	Controlled variable (also named manipulated variable). It is generated by the PID module with range [0.0 … 100.0] (default value = 0.0)
DTRK	BOOL	Digital Tracking Command (default value = FALSE)
TRK	REAL	Tracking signal [0…100.0] (default value = 0.0)
DAM	BOOL	Digital AutoMan Command (default value = FALSE)
UP	BOOL	Manual Up Command (default value = FALSE)
DOWN	BOOL	Manual Down Command (default value = FALSE)
AM_SL	REAL	Manual Delta Value (default value = 0.1)
DAUX	BOOL	Digital Auxiliary Command (default value = FALSE)
AUX	REAL	Auxiliary Signal [0…100.0] (default value = 0.0)

Output parameters

Label	Type	Description
OUT	REAL	Control Output [0.0 … 100.0] (default value = 0.0)
DBMP	BOOL	BUMPLESS command (default value = FALSE)

Description

This module is an Auto-Man station, to be connected downstream a PID module, providing the functionalities the operator needs, to interact with the loop. The CTRL_MV function block provides auto/manual/track/aux operating mode, and the commands for incrementally set the output. The operating modes are defined by a set of digital inputs, as listed above, ordered according to their priority level.

Trk *DTRK*

AutoMan *DAM*

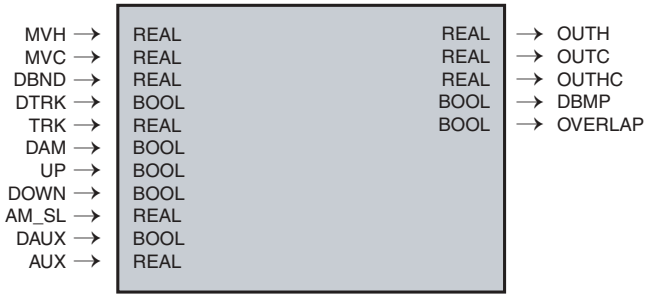
Aux *DAUX*

Depending on these digital inputs, the functionality of the module changes as follows:

- If none of these digital signals are active, the module doesn't alter the control output signal (from the PID module).
- If *DTRK* (Tracking) is active, the output *OUT* assumes the value of the analog input *TRK*.
- If the *DAM* input is active (and *DTRK* is inactive), the current outputs are not driven any longer according to the PID outputs, but they are changed only from the *UP* and *DOWN* commands. When the *DAM* and one of the two *UP* or *DOWN* commands are active, every cycle of the PLC the output of the function block moves (up or down) by the *AM_SL* quantity
- If *DAUX* (Aux) is active (and both *DAM* and *DTRK* are inactive), the output *OUT* assumes the value of the analog input *AUX*.

1-4-3 CTRL_HCMV

FB Prototype



Input parameters

Label	Type	Description
MVH	REAL	Analog input, corresponding to the control output (also named manipulated output) of the Heat PID module. Value in [0.0 … 100.0] (default value = 0.0)
MVC	REAL	Analog input, corresponding to the control output (also named manipulated output) of the Cool PID module. Value in [0.0 … 100.0] (default value = 0.0)
DBND	REAL	Deadband, with range from [-10.0 … +10.0] (default value = 0.0)
DTRK	BOOL	Digital Tracking Command (default value = FALSE)
TRK	REAL	Tracking signal [-100.0…100.0] (default value = 0.0)
DAM	BOOL	Digital AutoMan Command (default value = FALSE)
UP	BOOL	Manual Up Command (default value = FALSE)
DOWN	BOOL	Manual Down Command (default value = FALSE)
AM_SL	REAL	Manual Delta Value (default value = 0.1)
DAUX	BOOL	Digital Auxiliary Command (default value = FALSE)
AUX	REAL	Auxiliary Signal [-100.0…100.0] (default value = 0.0)

Output parameters

Label	Type	Description
OUTH	REAL	Control Output for HEAT [0.0 … 100.0] (default value = 0.0)
OUTC	REAL	Control Output for COOL [0.0 … 100.0] (default value = 0.0)
OUTHC	REAL	Control Output for visualization [-100.0 … 100.0] (default value = 0.0)
DBMP	BOOL	BUMPLESS command (default value = FALSE)
OVERLAP	BOOL	Active if both OUTH and OUTC are active (default value = FALSE)

Description

This module is an Output station for an Heat/Cool loop, providing the functionalities the operator needs to interact with the loop. The loop is controlled by two PIDs, connected downstream to this module; each one is dedicated to one channel. The module provides auto/manual/track/aux operating mode, and the commands for incrementally set the output. The operating modes, that are unique for both the Cool and the Heat output, are defined by a set of digital inputs, as listed above, ordered according to their priority level.

Trk *DTRK*

AutoMan *DAM*

Aux *DAUX*

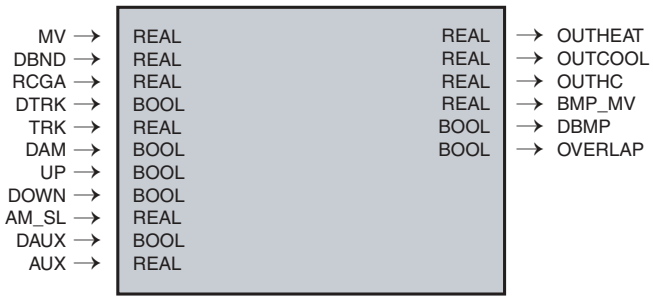
Depending on these digital inputs, the functionality of the module changes as follows:

- If none of these digital signals are active, the module does not alter the control output signals, that are led, unchanged, directly from the input *MVC* and *MVH* to the output *OUTH* and *OUTC*. The *DBND* parameter defines how the Cool and the Heat channel interact each other. When it is positive, this parameter defines the dead band width, centered around a value 0 of the output. It means that for each output value included in the deadband both the *OUTC* and the *OUTH* output are 0. When it is negative, the opposite occurs; for each value included in the dead band (now named cross band), both the *OUTC* and the *OUTH* are different from 0.
- If *DTRK* (Track) is active, the outputs are set according to the value of the analog input *TRK*. This signal has a range from *[-100.0 … +100.0]* and it drives both *OUTH* and *OUTC*. When its value is in the range -100 to 0, it means that a Cool action must be taken; therefore, *OUTH* is set to 0, while *OUTC* is set to the value of *TRK*, with the sign inverted. Inversely, when it is in the range 0 to 100, *OUTC* is set to 0, while *OUTH* is set to *TRK*. Please note that the tracking value is not affected by the *DBND* parameter.
- If the *DAM* input is active (and *DTRK* is inactive), the current outputs are not driven any longer according to the PID outputs, but they are changed only from the *UP* and *DOWN* commands. When the *DAM* and one of the two *UP* or *DOWN* commands are active, every cycle of the PLC the output of the function block moves (up or down) by the *AM_SL* quantity
- If *DAUX* (Aux) is active(and both *DAUX* and *DTRK* are inactive), the outputs are set according to the value of the analog input *AUX*. This signal has a range from *[-100.0 … +100.0]* and it drives both *OUTH* and *OUTC*. When its value is in the range -100 to 0, it means that a Cool action must be taken; therefore, *OUTH* is set to 0, while *OUTC* is set to the value of *AUX*, with the sign inverted. Inversely, when it is in the range 0 to 100, *OUTC* is set to 0, while *OUTH* is set to *AUX*. These values are valid if the *DBND* of the phase splitter is set to 0, otherwise they slightly change.

1-4-4 CTRL_SPLITMV

FB Prototype

CTRL_SPLITMV



Input parameters

Label	Type	Description
MV	REAL	Controlled variable (also named manipulated variable). It is generated by the PID module with range [0.0 … 100.0] (default value = 0.0)
DBND	REAL	Deadband, with range from [-10.0 … +10.0] (default value = 0.0)
RCGA	REAL	Relative Cool Gain (default value = 1.0)
DTRK	BOOL	Digital Tracking Command (default value = FALSE)
TRK	REAL	Tracking signal [0…100.0] (default value = 0.0)
DAM	BOOL	Digital AutoMan Command (default value = FALSE)
UP	BOOL	Manual Up Command (default value = FALSE)
DOWN	BOOL	Manual Down Command (default value = FALSE)
AM_SL	REAL	Manual Delta Value (default value = 0.1)
DAUX	BOOL	Digital Auxiliary Command (default value = FALSE)
AUX	REAL	Auxiliary Signal [0…100.0] (default value = 0.0)

Output parameters

Label	Type	Description
OUTHEAT	REAL	Control Output for HEAT [0.0 … 100.0] (default value = 0.0)
OUTCOOL	REAL	Control Output for COOL [0.0 … 100.0] (default value = 0.0)
OUTHC	REAL	Control Output for visualization [-100.0 … 100.0] (default value = 0.0)
BMP_MV	REAL	BUMPLESS output value (default value = 0.0)
DBMP	BOOL	BUMPLESS command (default value = FALSE)
OVERLAP	BOOL	Active if both OUTH and OUTC are active (default value = FALSE)

Description

This module is an Output station for an Heat/Cool loop, providing the functionalities the operator needs to interact with the loop. The loop is controlled by one PID, connected downstream to this module; when the PID output value is between [50.0… 0.0] the cool action is active, when the PID output value is between [50.0… 100.0] the heat action is active. The function block applies also a factor to the cool action called Relative Cool Gain – *RCGA*. The module provides auto/manual/track/aux operating mode, and the commands for incrementally set the output.

The operating modes, that are unique for both the Cool and the Heat output, are defined by a set of digital inputs, as listed above, ordered according to their priority level.

Trk *DTRK*

AutoMan *DAM*

Aux *DAUX*

Depending on these digital inputs, the functionality of the module changes as follows:

- If none of these digital signals are active, the module split the input *MV* into the two action *MVH* and *MVC*. The *DBND* parameter defines how the Cool and the Heat channel interact each other. When it is positive, this parameter defines the dead band width, centered around the value 50.0 of the input. It means that when the input is in the range [50.0 – DBND… 50.0 + DBND] both the *OUTC* and the *OUTH* output are 0. When it is negative, the opposite occurs; for a value included in the dead band, both the *OUTC* and the *OUTH* are different from 0.
- If *DTRK* (Track) is active, the outputs are set according to the value of the analog input *TRK*. This signal has a range from [0.0 … 100.0] and it drives both *OUTH* and *OUTC*. When its value is in the range 0.0 to 50.0, it means that a Cool action must be taken; therefore, *OUTH* is set to 0, while *OUTC* is set to *[RCGA * 2.0 * (50.0 - TRK)]*. Inversely, when it is in the range 50.0 to 100, *OUTC* is set to 0, while *OUTH* is set to *[2.0 * (TRK – 50.0)]*. Please note that the tracking value is not affected by the *DBND* parameter.
- If the *DAM* input is active (and *DTRK* is inactive), the current outputs are not driven any longer according to the PID output, but they are changed only from the *UP* and *DOWN* commands. When the *DAM* and one of the two *UP* or *DOWN* commands are active, every cycle of the PLC the output of the function block moves (up or down) by the *AM_SL* quantity
- If *DAUX* (Aux) is active (and both *DAUX* and *DTRK* are inactive), the outputs are set according to the value of the analog input *AUX*. This signal has a range from [0.0 … 100.0] and it drives both *OUTH* and *OUTC*. When its value is in the range 0.0 to 50.0, it means that a Cool action must be taken; therefore, *OUTH* is set to 0, while *OUTC* is set to *[RCGA * 2.0 * (50.0 - AUX)]*. Inversely, when it is in the range 50.0 to 100, *OUTC* is set to 0, while *OUTH* is set to *[2.0 * (AUX – 50.0)]*. These values are valid if the *DBND* of the phase splitter is set to 0, otherwise they slightly change.

1-4-5 CTRL_SRV

FB Prototype

Input parameters

Input	Type	Description
MV	REAL	Required position [%] [admitted range 0.0 … 100.0] (default value = 0.0)
TTRAVEL	REAL	Servomotor travel time [s] [admitted range 1.0 … 600.0] (default value = 60.0)
DBND	REAL	Dead band [%] [admitted range 0.0 … 100.0] (default value = 2.0)
MINRES	BOOL	Band value used to consider the target reached [%] [admitted range 0.0 … 100.0] (default value = 0.2)
INITPOS	REAL	Init position [%] [admitted range 0…100.0] (default value = 0.0)
DLY	REAL	Timeout value [s] (default value = 0.1)
RESETPOS	BOOL	Reset Command (default value = FALSE)
OUTMODE	BOOL	Output Command (default value = TRUE)

Output parameters

Label	Type	Description
OPEN	BOOL	Open valve command (default value = FALSE)
CLOSE	BOOL	Close valve command (default value = FALSE)
POS	REAL	Calculated valve position [%] [range 0…100.0] (default value = 0.0)

Description

This module generates the valve *OPEN* and *CLOSE* commands in relation to the position required by the MV signal and to the internal valve position. This value is calculated using the *TTRAVEL* value and the sampling time of the task where this function block is used. The *OPEN* & *CLOSE* commands are set according to *MINRES* and *DBND* parameters.

The *DLY* parameter specify a timeout that has to be waited before to move the valve from an already reached position to a new desired one.

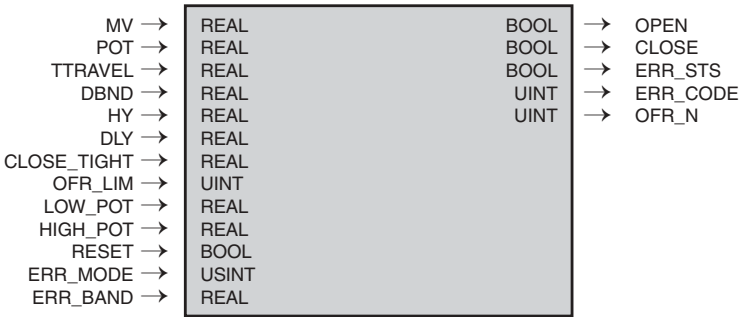
The *OUTMODE* command specify if the *OPEN* or *CLOSE* command must be maintained when the valve reaches the 0.0 or 100.0 position. If *OUTMODE* is *TRUE* the command is maintained.

The *RESETPOS* command perform a reset of the function block code, and assign to the POS output the *INITPOS* value.

1-4-6 CTRL_SRV_POS

FB Prototype

CTRL_SRV_POS



Input parameters

Input	Type	Description
MV	REAL	Required target position [%] [admitted range 0.0 … 100.0] (default value = 0.0)
POT	REAL	Potentiometer position value [%] [admitted range 0.0 … 100.0]
TTRAVEL	REAL	Servomotor travel time [s] [admitted range 0.1 … 600.0] (default value = 60.0)
DBND	REAL	Dead band [%] [admitted range 0.0 … 100.0] (default value = 0.5)
HY	REAL	Band value used to consider the target reached [%] [admitted range 0.0 … 100.0] (default value = 0.2)
DLY	REAL	Delay between 2 opposite commands [s] [admitted range 0.1 … 100.0] (default value = 0.1)
CLOSE_TIGHT	REAL	Output maintained band [%] [admitted range 0.0 … 100.0] (default value = 0.0)
OFR_LIM	UINT	Potentiometer Out of Range maximum occurrences limit [num] [admitted range 0 … 65535] (default value = 0)
LOW_POT	REAL	Potentiometer Low limit [num] [admitted range 0 … 65535] (default value = 0.0)
HIGH_POT	REAL	Potentiometer High limit [num] admitted range 0 … 65535] (default value = 100.0)
RESET	BOOL	Error conditions Reset (default value = FALSE)
ERR_MODE	USINT	Outputs behavior in Error mode [num] [admitted range 0 … 2] (default value = 0)
ERR_BAND	REAL	Out of Range Error band [%] [admitted range 0.0 … 100.0] (default value = 0.0)

Output parameters

Output	Type	Description
OPEN	BOOL	Open valve command (initial value = FALSE)
CLOSE	BOOL	Close valve command (initial value = FALSE)
ERR_STS	BOOL	Error condition (initial value = FALSE)
ERR_CODE	UINT	Error code, bit mask [0 …7] (initial value = 0)
OFR_N	UINT	Out of Range occurrences, value [0 …65535](initial value = 0)

Description

This FB provide a servomotor positioner control in close loop mode with 2 digital commands (*OPEN* and *CLOSE*). The FB receives the percentage desired position (*MV*) and activates the digital commands if the actual position (*POT*) differs by the dead band (*DBND*). In order to properly calculates a correct position value in per-

centage, it is necessary to set the scale limits (*LOW_POT* and *HIGH_POT*). The target position is considered reached when the actual value is within the *HY* band.

The *CLOSE_TIGHT* parameter define a deviation band on the 0.0 and 100.0% value: when the *MV* will be within these areas, the digital commands OPEN and *CLOSE* will be maintained forced. The function is disabled with a value of 0.0 %.The *DLY* parameter specify a time between two opposite commands when a position has been already reached.

The *RESET* command perform a reset of the function block when in *ERROR* mode: ***THE FB REMAIN IN THIS STATE UNTIL A RESET COMMAND IS PERFORMED!!!***

ERR_MODE

Outputs behaviour in case of error:

ERR_MODE value	Control Output Status	
0	Open = FALSE	Close = FALSE
1	Open = TRUE	Close = FALSE
2	Open = FALSE	Close = TRUE

ERR_BAND

Outside band of the MV range (0.0...100.0%) where to evaluate the error conditions.

ERR_CODE

Bits mask associated to the ERR_CODE value:

ERR_CODE bit	Control Output Status
Bit 1	Validation of input POT value (LOW_POT and HIGH_POT) exceeded OFR_LIM value
Bit 2	MV out of range
Bit 3	Timeout to reach the target value (in according to travel time and actual position)

1-4-7 CTRL_TPO

FB Prototype

CTRL_TPO

Input parameters

Label	Type	Description
MV	REAL	Required duty cycle [0.0 … 100.0] (default value = 0.0)
TCYCLE	REAL	Cycle time in seconds (default value = 20.0)
TMIN	REAL	Minimum time is seconds (default value = 0.0)

Output parameters

Label	Type	Description
DOUT	BOOL	Time proportional output (default value = FALSE)
ALARM	BOOL	Alarm flag (default value = FALSE)

Description

This module creates a time proportional output requiring as inputs the duty cycle value (*MV*), and the period of the signal (*TCYCLE*). The *TMIN* input specifies the minimum pulse length that can be generated by the function block. If the required duty cycle sets a pulse length less then *TMIN*, the function block generates a pulse of TMIN, and it modifies the period of the signal in order to maintain the duty cycle passed by *MV*. In this case the output *ALARM* will be active. The same situation occurs if the passed duty cycle is grater then *TCYCLE* – *TMIN*: in this case the function block modifies the length of the generated pulse in order to maintain the duty cycle at the required value, and it forces the output to be FALSE for a time equal to *TMIN*. Also in this case the output *ALARM* will be active.

Note: ASCON recommends the use of this function block in a cyclic task inside the OpenPCS programming tool.

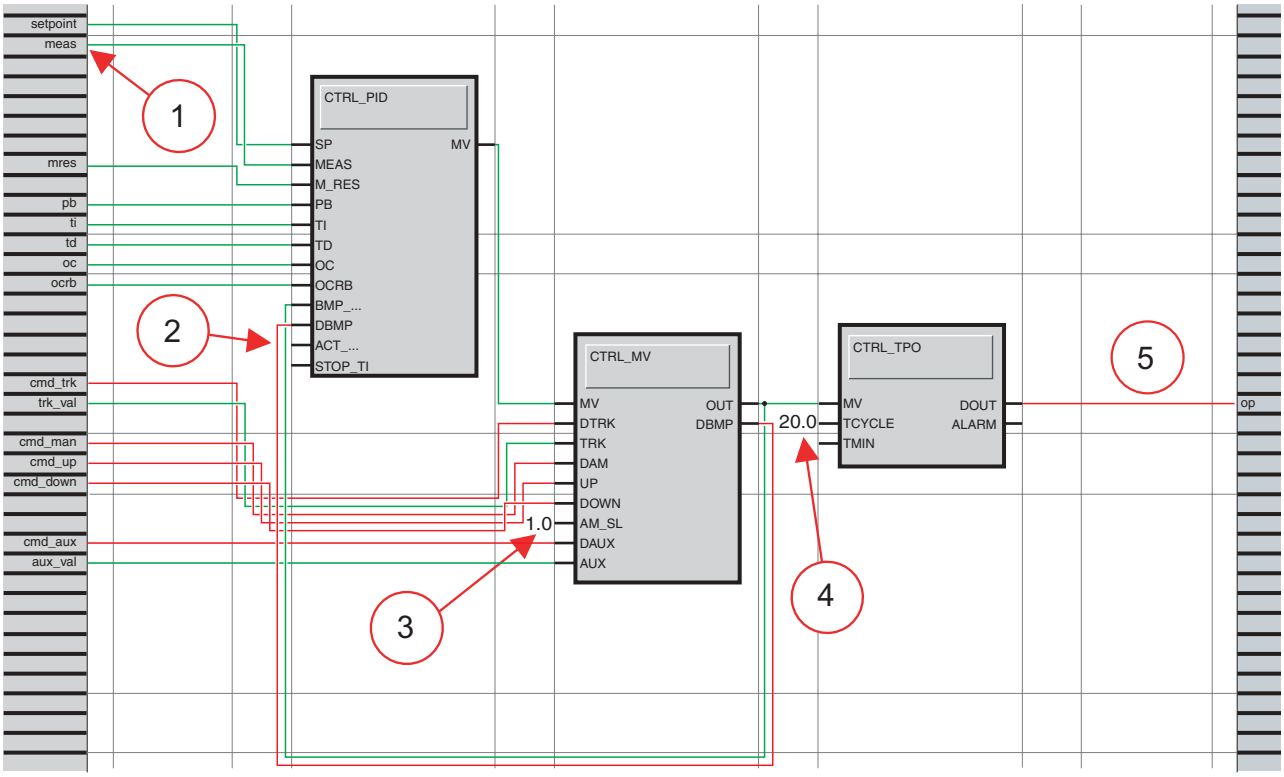
1-5 Example task

The following is a simple task example in which a proportional time output control is implemented. In this case the used Function Blocks are:

- CTRL_PID for the control,
- CTRL_MV for manual automatic station and tracking and aux commands,
- CTRL_TPO for the particular used output.

Remarks:

1. "*setpoint*" and "*meas*" variables must be already scaled in the range [0.0 … 100.0],
2. the chosen PID action is the REVERSE action which is default for the Function Block (the corresponding input has not been assigned),
3. for manual command the output variation speed has been set to one unit for every task work cycle,
4. the proportional signal time has been set to 20 seconds.



Chapter 2
MODBUS TCP/IP

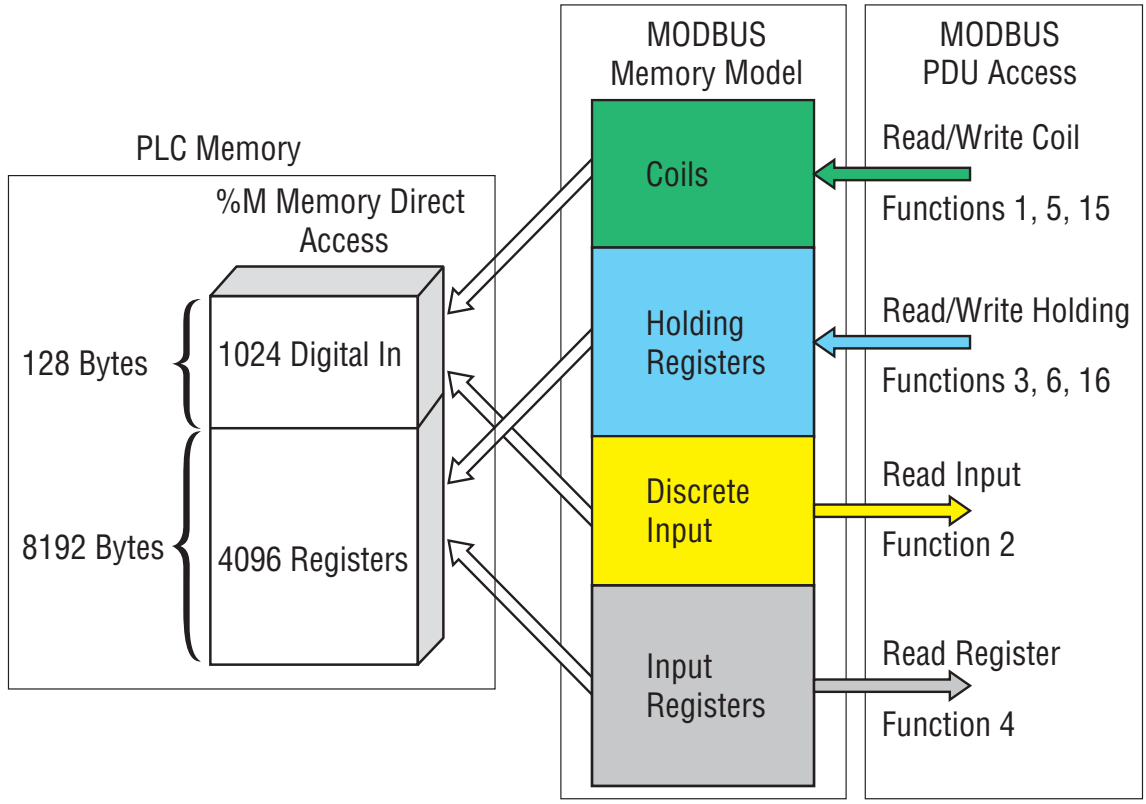
The Ascon CPU modules implement a Modbus TCP/IP Server agent which can be configured and accessed in the OpenPCS programming environment. It is also possible to partially configure the agent during the configuration session run when the CPU is started up with the VT100 terminal. The Modbus TCP/IP agent can be accessed externally over the Ethernet network via the unit 10 base T port. In the system FW, the Modbus TCP/IP protocol activities have an execution interval of 100 ms. This means, for instance, that a client running a query must allow for a response timeout of no less than 100 ms. The Modbus TCP/IP agent can handle up to 10 TCP/IP connections at a time. It is also possible to set a parameter which determines the maximum number of processable messages per cycle, so that any additional messages are processed by the next cycle. The user can also define two groups of IP addresses. The first group is called the “priority connection pool”, which constitutes the set of addresses whose connections will not be terminated when the system receives a request for a new connection and the 10 available connections are already all occupied. The second group is the “security address pool”, which is the set of addresses that can access the Modbus TCP/IP Server agent when security functions are enabled. Among all settable parameters, one sets the time the system waits before automatically closing an idle connection.

2-1 Modbus TCP/IP Server agent

The Modbus TCP/IP Server agent resides in the system Firmware and must be activated and configured by the user using a number of Function blocks and initial configuration session menus which are described below. Once the server is configured, the agent remains operative and grants a client access to specific memory areas shared with the IEC61131-3 programming environment. The Modbus TCP/IP Server agent shares the memory area with Modbus serial slave agent #2, which means that the two agents cannot be used at the same time. The available modbus functions and the memory map shared by the IEC61131-3 environment and communication protocol are shown below.

				Function Codes		
				Code	**Sub code**	**(hex)**
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02
		Internal Bits Or Physical Coils	Read Coils	01		01
			Write Single Coil	05		05
			Write Multiple Coils	15		0F
	16 bit access	Physical Input Registers	Read Input Register	04		04
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03
			Write Single Register	06		06
			Write Multiple Registers	16		10
Diagnostics			Read Exception status	07		07
			Diagnostics	08	0000	08

The protocol's support data area follows the Modbus native memory model:



This memory model, accessible via the modbus TCP/IP Server agent, is mapped on a memory zone which can be accessed via the IEC61131-3 programming environment as shown in the following table:

Protocol Agent	Protocol Memory Areas	PLC Memory Mapping
ModBus TCP/IP Server	Coils	%M10000.0 to %M10127.7
	Holding Registers	%MW10128.0 to %MW18318.0
	Discrete Inputs	%M10000.0 to %M10127.7
	Input Registers	%MW10128.0 to %MW18318.0
	Status Register	%MB18319.0

The following tables show the correspondence between the modbus addresses of the registers and bits accessible via the IEC61131-3 protocol and the syntax used to access the same memory areas.

COILS Modbus TCP/IP Server

MODBUS Address	IEC61131-3 SYNTAX
0	%MX10000.0
1	%MX10000.1
2	%MX10000.2
3	%MX10000.3
4	%MX10000.4
5	%MX10000.5
6	%MX10000.6
7	%MX10000.7
8	%MX10001.0
9	%MX10001.1
...	...
1017	%MX10127.1
1018	%MX10127.2
1019	%MX10127.3
1020	%MX10127.4
1021	%MX10127.5
1022	%MX10127.6
1023	%MX10127.7

REGISTERS Modbus TCP/IP Server

MODBUS ADDRESS	IEC61131-3 SYNTAX
0	%MW10128.0
1	%MW10130.0
2	%MW10132.0
3	%MW10134.0
4	%MW10136.0
5	%MW10138.0
6	%MW10140.0
7	%MW10142.0
8	%MW10144.0

9	%MW10146.0
...	...
2048	%MW14224.0
2049	%MW14226.0
2050	%MW14228.0
2051	%MW14230.0
2052	%MW14232.0
...	...
4088	%MW18304.0
4089	%MW18306.0
4090	%MW18308.0
4091	%MW18310.0
4092	%MW18312.0
4093	%MW18314.0
4094	%MW18316.0
4095	%MW18318.0

Once the Server agent has been configured, it runs in the background and automatically updates the above indicated memory areas. A function block is also available to check the status of each separate TCP/IP connection: **MB_TCP_CONN_STATUS**.

2-1-1 ENABLE_MODBUS_TCP_SERVER

Function Block setting and activating the Modbus TCP/IP Server agent.

FB Prototype

ENABLE_MODBUS_TCP_SERVER

Input description

Label	Type	Description	Range
SERVER_PORT	UINT	Port on which the server listens	
USE_SECURITY	BOOL	Use of security functions	TRUE/FALSE
TIMEOUT	UINT	Timeout in units of 100 ms*	0..65535

Note: * if timeout = 0 the timeout computation function is disabled

Output description

Label	Type	Description	Range
Oprs	USINT	Execution Result	**0:** Ok **1:** Server not available **2:** Resources not available **4:** TCP/IP Problems

Description

This FB sets the initialisation and configuration parameters of the Modbus TCP/IP Server agent. After execution of the Function Block, if successful, the agent is operative and responds to any message it receives. The “SERVER_PORT” parameter tells the agent on which port to listen while waiting for the TCP/IP connections. The default setting is 502 for the Modbus TCP/IP protocol. The agent is always set to listen on the default port. With the “SERVER_PORT” parameter the user can also choose a second port on which the agent is set to listen. The timeout parameter indicates the agent the maximum interval between two consecutive queries. If this interval is exceeded, the agent does not change its operation, but simply flags the timeout expiry the next time the **MB_TCP_CONN_STATUS** Function Block is called. To evaluate the timeout, the counter starts at the end of the execution of current Function Block and is reset every time a new query is received.

2-1-2 CLOSE_MODBUS_TCP_SERVER

Function Block disabling the Modbus TCP/IP Server.

FB Prototype

CLOSE_MODBUS_TCP_SERVER

This function disables the Modbus TCP/IP Server agent execution and makes the resources newly available for a new initialisation or for the Modbus RTU Slave 2.

2-1-3 MB_TCP_CONN_STATUS

Function Block showing the status of a Modbus TCP/IP connection.

FB Prototype

MB_TCP_CONN_STATUS

Input description

Label	Type	Description	Range
CONNECTION	USINT	Connection whose status is to be assessed	0...9

Output description

Label	Type	Description	Range
Status	USINT	Communication Status	**0:** No Operations **1:** Memory value updated **2:** Timeout expired **3:** Agent error
Oprs	USINT	Execution Result	**0:** Ok **1:** Server not available **2:** Timeout expired **4:** Inactive connection **8:** Protocol exception **10:** No connection

Description

This Function Block shows the communication status on a specified connection. As noted above, the system can initiate up to 10 connections at a time. Current FB acquires a number from 0 to 9 as its input, which indicates the connection whose status is to be questioned. Naturally, one of the possible status is “inactive connection”, i.e. no connection has been initiated in the requested position. Other Function Blocks, described later, make it possible to trace the IP addresses and connection port from the position, and vice-versa.

2-1-4 MB_TCP_GET_CONN_BY_ADDR

Function Block returning the position of a connection identified by the client IP address, the client port and the server port.

FB Prototype

MB_TCP_GET_CONN_BY_ADDR

Input description

Label	Type	Description	Range
IP_ADDR	STRING	Client IP address	
CLIENT_PORT	UINT	Client port	
SERVER_PORT	UINT	Server port	

Output description

Label	Type	Description	Range
Connection	USINT	Communication Position	0...9
Oprs	USINT	Execution Result	**0:** Ok **1:** Server not available **2:** Invalid IP address **4:** Inactive connection

Description

This Function Block indicates the presence of the connection uniquely defined by three input parameter values in one of the 10 available positions.

2-1-5 MB_TCP_GET_CONN_CONFIG

Function Block reporting the characteristics of an active connection in one of the 10 available positions.

FB Prototype

MB_TCP_GET_CONN_CONFIG

Input description

Label	Type	Description	Range
CONNECTION	USINT	Connection whose configuration is to be recovered	0...9

Output description

Label	Type	Description	Range
IP_ADDR	STRING	Client IP address	
CLIENT_PORT	UINT	Client port	
SERVER_PORT	UINT	Server port	
Oprs	USINT	Execution Result	**0:** Ok **1:** Server not available **2:** No connection

Description

This Function Block indicates whether the position specified by the input connection is currently occupied by an active connection or not. Furthermore, if the connection is active, it reports the three values which uniquely identify it, namely: Client IP address and port and Server port.

2-1-6 MB_TCP_CLOSE_CONN

Function block closing one of the 10 active connections.

FB Prototype

MB_TCP_CLOSE_CONN

Input description

Label	Type	Description	Range
CONNECTION	USINT	Connection to be closed	0..9

Output description

Label	Type	Description	Range
Oprs	USINT	Execution Result	**0:** Ok **1:** Server not available **2:** No connection **4:** Inactive connection

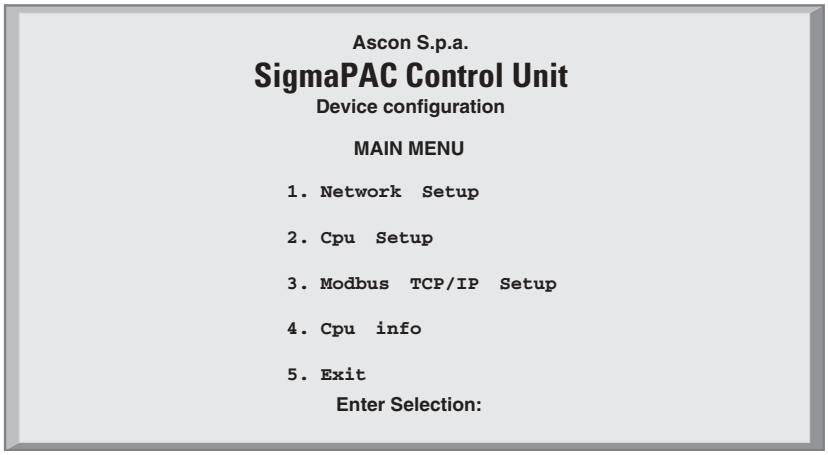
Description

This Function Block closes the connection associated with one of the 10 available positions.

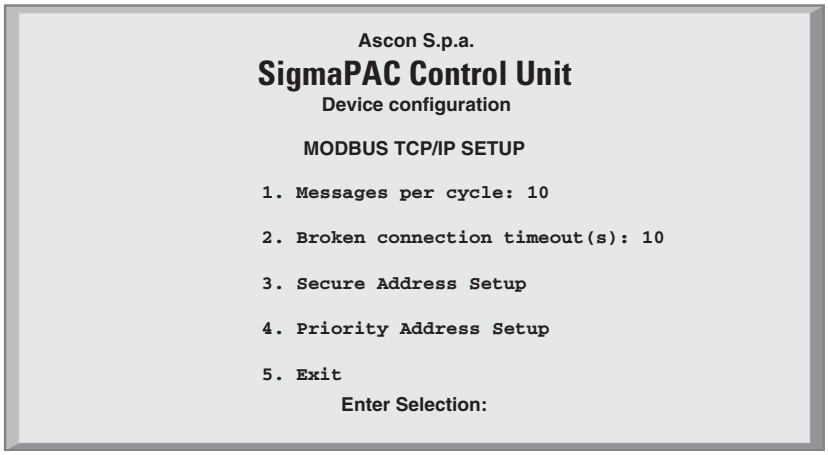
2-1-7 Configurations using the Terminal VT100 initial setup session

The VT100 terminal is used in the initial configuration session to specify the group of addresses making up the “priority connection pool” and the “security address pool”. Two other parameters, controlling the execution of the Modbus TCP/IP Server agent, can also be configured: the number of messages processable per cycle and the TCP connection idle time after which the connection is closed.

Note: The Control Unit configuration session is described in the “*User Manual*” specific for the CPU model in use.
The main configuration menu is shown in the picture below:



The menu shown in the following picture is displayed, if item 3 is selected.



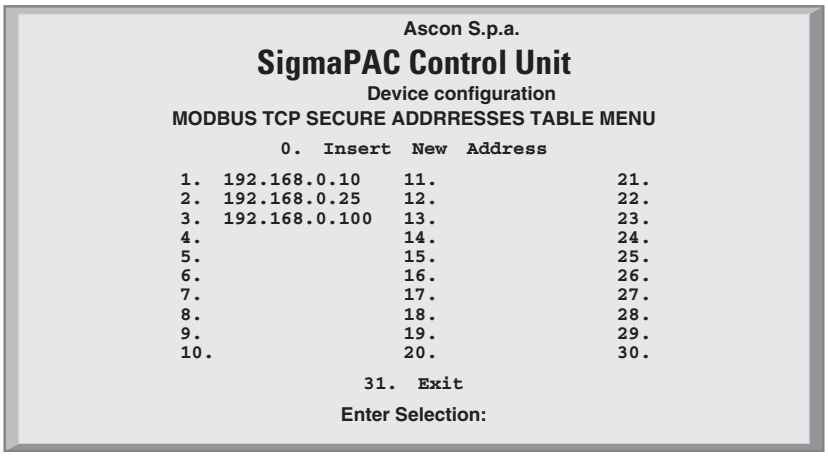
Item 1 specifies the number of messages processed per cycle. The value entered must be within the range of 1 to 50.

Item 2 specifies the connection idle timeout. The value entered must be within the range of 10s to 5400s. The connection probe in case of long communication idle time complies with the “keep alive” protocol provided by TCP/IP specifications. This process performs the following operation sequence:

1. The system resets an internal idle timeout when it receives a new message.
2. If the internal idle timeout reaches the value set in point 2, the system sends a so-called “probe” message which checks whether the connection is still active.
3. If the probe message receives a response, the connection is active and the idle timeout is reset.
4. If the probe message does not receive a response within 10 seconds, the system sends a new "probe" message.
5. If the probe message receives a response, the idle timeout is reset.

6. The system sends up to 4 probe messages at intervals of 10 seconds, and if none of them receives a response, the connection is closed.

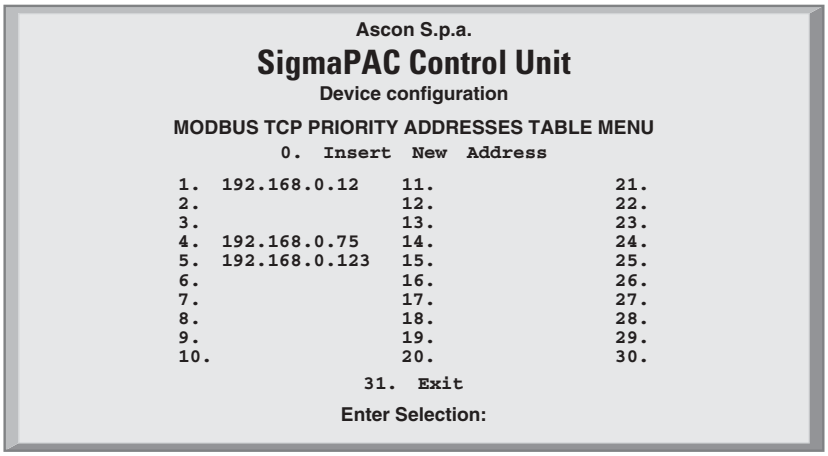
The page shown in the following picture is displayed, if menu item 3 is selected.



This menu specifies the addresses which can access the Modbus TCP/IP Server agent when security functions are enabled. To enter a new address, select item 0 and then enter the address, which is entered in the first free position out of the 30 available. To delete an address, select the number which it is associated with.

As described above, an address list called “priority connection pool” is also available, which can be configured by selecting the menu item 4 in the “MODBUS TCP IP SETUP MENU”. The page shown in the following picture is thus displayed.

The address entry process is the same as that described for the entry of the “security address pool” addresses.



The addresses in the “priority connection pool” are given priority by the system: as explained above, up to 10 TCP connections can be maintained at the same time by the Modbus TCP/IP Server agent. When all available connections are occupied, the system must make a choice when a new connection is requested. Based on the principle that every new connection request must be satisfied, a process must be implemented which selects the connection to be closed to make room for a new one. The first candidates for disconnection are all those addresses which DO NOT belong to the “priority connection pool”. The selection is further refined by identifying the connection which has been longest inactive.

2-1-8 Application example of the Modbus TCP/IP Server agent

This section describes the code of a simple application which initialises the Modbus TCP/IP Server agent and periodically increments the value of 4 registers which can be read by a client. The example also shows the call to other Function Blocks controlling the agent.

2-1-9 Global application variables

The program tasks share a number of global variables which are code-declared as follows:

```
VAR_GLOBAL
     init_port_mb_tcp: BOOL := true;
     error_mb_tcp: BOOL;
END_VAR
```

The variables init_port_mb_tcp and error_mb_tcp check that the initialisation Function Blocks are only called once, while the others are called only if the initialisation has terminated successfully.

2-1-10 HW port initialisation

This task initialises the agent. The task code variable declaration and executable code are shown below:

Variable declaration

```
VAR_EXTERNAL

     init_port_mb_tcp : BOOL;
     error_mb_tcp     : BOOL;

END_VAR

VAR_GLOBAL

END_VAR

VAR

     mb_tcp_oprs      : USINT;
     init_mb_tcp             : ENABLE_MODBUS_TCP_SERVER;
     close_mb_tcp     : CLOSE_MODBUS_TCP_SERVER;

END_VAR
```

Task code

```
if init_port_mb_tcp then

     init_port_mb_tcp := false;
     error_mb_tcp := FALSE;

     close_mb_tcp();

     init_mb_tcp(
     SERVER_PORT:= 503,
     USE_SECURITY:= 0,
     TIMEOUT:= 10
     |
     mb_tcp_oprs := OPRS);

     if mb_tcp_oprs > 0 then
          error_mb_tcp := true;
     end_if;

end_if;
```

Note how, before enabling the agent, the Function Block is called to close it. As far as the logic sequence of agent control is concerned, this might seem incorrect, but the open functions have been designed to return an error if the port is already busy. Consequently, if the agent is not disabled before downloading a new version of the firmware, the agent will remain occupied and the system will not allow it to be used when the program is launched.

2-1-11 Example code

This task works with the registers which are addressed by the Modbus TCP/IP Server agent.

Variable declaration

```
VAR_EXTERNAL

     init_port_mb_tcp   : BOOL;
     error_mb_tcp       : BOOL;

END_VAR

VAR_GLOBAL

END_VAR

VAR

     oprs               :USINT;
     status             :USINT;
     conn               :USINT;
     client_port        :UINT := 1500;
     server_port        :UINT := 503;
     connection         :USINT:= 0;
     ip_addr            :STRING(16) := '192.168.5.1';
     connected          :BOOL;

     con_from_addr      :BOOL := false;
     conn_from_addr     :MB_TCP_GET_CONN_BY_ADDR;

     con_config         :BOOL := false;
     conn_config        :MB_TCP_GET_CONN_CONFIG;

     con_status         :BOOL := false;
     connection_stat    :MB_TCP_CONN_STATUS;

     close_con          :BOOL := false;
     close_conn         :MB_TCP_CLOSE_CONN;

     at %MW10128.0      :UINT;
     at %MW10130.0      :UINT;
     at %MW10132.0      :UINT;
     at %MW10134.0      :UINT;

END_VAR
```

Task code

```
if init_port_mb_tcp = false then
     if error_mb_tcp = false then

          %MW10128.0 := %MW10128.0 + 1;
          %MW10130.0 := %MW10130.0 + 2;
          %MW10132.0 := %MW10132.0 + 3;
          %MW10134.0 := %MW10134.0 + 4;


          if con_from_addr = true then
               conn_from_addr(
                                IP_ADDR := ip_addr,
                                CLIENT_PORT := client_port,
                                SERVER_PORT := server_port
                                |
                                conn := CONNECTION,
                                oprs := OPRS);
               con_from_addr := false;
          end_if;

          if con_config = true then
               conn_config(
                                CONNECTION := connection
                                |
                                connected := CONNECTED,
                                client_port := CLIENT_PORT,
                                server_port := SERVER_PORT,
                                ip_addr := IP_ADDR,
                                oprs := OPRS);
               con_config := false;
          end_if;

          if con_status = true then
               connection_stat(
                                CONNECTION := connection
                                |
                                status := STATUS,
                                oprs := OPRS);
               con_status := false;
          end_if;

          if close_con = true then
               close_conn(
                                CONNECTION := connection
                                |
                                oprs := OPRS);
               close_con := false;
          end_if;

     end_if;
end_if;
```

Chapter 3
Serial communications

The Ascon CU-02 and MP-01 units are IEC61131-3 programmable devices with several communication ports: 10baseT ethernet, serial RS232/RS485 and a CAN port in case of CU-02 unit. The available serial ports can be used for applications requiring Modbus communication. Two slave agents and one master agent have been implemented; the Modbus agents can be associated to each of the physical available ports. An agent has also been implemented for an ASCII string communication; this one can only be associated to the serial service port (X0 for MP-01 and X1 for CU-02). The thread in the FW executing all serial communication activities repeats the operations every 100 ms. This means that in case for example of a serial port configuration with Modbus Slave protocol, a possible master executing a query must consider a response timeout not lower than 100 ms.

3-1 Serial communication configuration

The serial communication configuration of the auxiliary ports can be made using two Function Blocks:

- OPEN_SERIAL_COMM
- CLOSE_SERIAL_COMM

These calls are used to define the configuration and the protocol to be assigned to the port. Two slave and one master protocols and an agent can be assigned for simple serial communication of ASCII strings. The 3 MODBUS and the ASCII protocols can be freely assigned to the ports. The following numbers classify the physical ports for the CU-02 and the MP-01 units:

CU-02	
0	RS232 port (X1)
1	RS232 port (X3)
2	RS485 port (X4)

MP-01	
0	RS232/RS485 port (X0)
1	RS485 port (X1)

The following numbers classify the protocols that are valid for both CU-02 and MP-01:

0. ASCII serial communication
1. Slave 1
2. Slave 2
3. Master

3-1-1 OPEN_SERIAL_COMM

Function Block enabling to initialise a port.

FB Prototype

OPEN_SERIAL_COMM

Input description

Label	Type	Description	Range
Port	USINT	**CU-02** Serial port selection variable	0, 1, 2
		MP-01 Serial port selection variable	0, 1
Baud	USINT	Communication baud rate (see table)	0, 1, 2, 3, 4, 5, 6
Parity	USINT	Parity bit configuration	**0:** none; **1:** Even; **2:** Odd
Data	USINT	No. of bits	7,8
Stop	USINT	Number of stop bits	1,2
Protocol	USINT	Port protocol	**0:** Serial ASCII **1:** Slave1, **2:** Slave2, **3:** Master

Output description

Label	Type	Description	Range
Oprs	USINT	Execution Result	**0:** Ok, **1:** port not available, **2:** protocol not available, **3:** invalid configuration, **4:** non-existent protocol

Description

This function configures the serial communication port and also sets the protocol to be used on it. The following table links the baud rate value to be entered in the function block with the set baud rate.

Value	Baudrate
0	2400
1	4800
2	9600
3	19200
4	38400
5	57600
6	115200

3-1-2 CLOSE_SERIAL_COMM

Function Block closing the serial port.

FB Prototype

CLOSE_SERIAL_COMM

Input description

Label	Type	Description	Range
Port	USINT	**CU-02** Serial port selection variable	0,1, 2
		MP-01 Serial port selection variable	0,1

Description

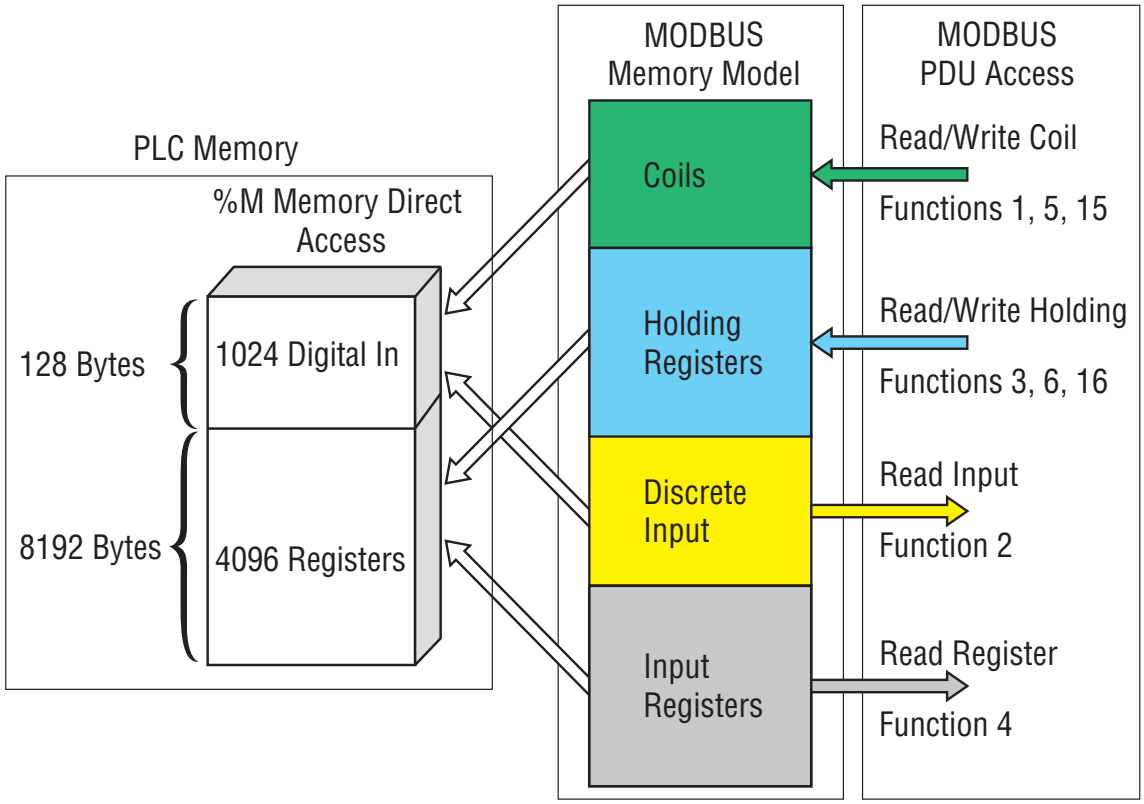
This function closes the serial communication port and releases its association with the previously assigned serial protocol. After the function block has been executed, the port is available for later use through the open function block.

3-2 Modbus Slave protocol

Once the Modbus Slave protocol is assigned to a port as described in function block OPEN_SERIAL_COMM, it is not operative yet, but needs to be configured. This is done with function block MODBUS_SLAVE_SETTINGS which assigns a Modbus network address and a timeout to the slave port. The system monitors the time between two consecutive queries to the slave and checks that such time is shorter than timeout. Once the slave is configured, the agent stays operative and grants a Master access to specific memory areas shared with the IEC61131-3 programming environment. The Modbus agent responds to the following functions and processes one message approximately every 100 ms.

				Function Codes		
				Code	Sub code	(hex)
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	02		02
		Internal Bits Or Physical Coils	**Read Coils**	01		01
			Write Single Coil	05		05
			Write Multiple Coils	15		0F
	16 bit access	Physical Input Registers	**Read Input Register**	04		04
		Internal Registers Or Physical Output Registers	**Read Holding Registers**	03		03
			Write Single Register	06		06
			Write Multiple Registers	16		10
Diagnostics			**Read Exception status**	07		07
			Diagnostics	08	0000	08

The protocol's support data area complies with the Modbus native memory model:



This memory model, which is accessible via the Modbus Slave agent, is mapped on a memory zone which can be accessed via the IEC61131-3 programming environment as showed in the following table.

Protocol Agent	Protocol Memory Areas	PLC Memory Mapping
ModBus Slave Channel #1	Coils	%M1000.0 to %M1127.7
	Holding Registers	%MW1128.0 to %MW9318.0
	Discrete Inputs	%M1000.0 to %M1127.7
	Input Registers	%MW1128.0 to %MW9318.0
	Status Register	%MB9320.0
ModBus Slave Channel #2	Coils	%M10000.0 to %M10127.7
	Holding Registers	%MW10128.0 to %MW18318.0
	Discrete Inputs	%M10000.0 to %M10127.7
	Input Registers	%MW10128.0 to %MW18318.0
	Status Register	%MB18320.0

The following tables show the correspondence between the modbus addresses of the registers and bits accessible via the IEC61131-3 protocol and the syntax used to access the same memory areas.

COILS Slave 1

MODBUS Address	IEC61131-3 SYNTAX
0	%MX1000.0
1	%MX1000.1
2	%MX1000.2
3	%MX1000.3
4	%MX1000.4
5	%MX1000.5
6	%MX1000.6
7	%MX1000.7
8	%MX1001.0
9	%MX1001.1
...	...
1017	%MX1127.1
1018	%MX1127.2
1019	%MX1127.3
1020	%MX1127.4
1021	%MX1127.5
1022	%MX1127.6
1023	%MX1127.7

COILS Slave 2

MODBUS Address	IEC61131-3 SYNTAX
0	%MX10000.0
1	%MX10000.1
2	%MX10000.2
3	%MX10000.3
4	%MX10000.4
5	%MX10000.5
6	%MX10000.6
7	%MX10000.7
8	%MX10001.0
9	%MX10001.1
...	...
1017	%MX10127.1
1018	%MX10127.2
1019	%MX10127.3
1020	%MX10127.4
1021	%MX10127.5
1022	%MX10127.6
1023	%MX10127.7

REGISTERS Slave 1

MODBUS Address	IEC61131-3 SYNTAX
0	%MW1128.0
1	%MW1130.0
2	%MW1132.0
3	%MW1134.0
4	%MW1136.0
5	%MW1138.0
6	%MW1140.0
7	%MW1142.0
8	%MW1144.0
9	%MW1146.0
...	...
2048	%MW5224.0
2049	%MW5226.0
2050	%MW5228.0
2051	%MW5230.0
2052	%MW5232.0
...	...
4088	%MW9304.0
4089	%MW9306.0
4090	%MW9308.0
4091	%MW9310.0
4092	%MW9312.0

4093	%MW9314.0
4094	%MW9316.0
4095	%MW9318.0

REGISTERS REGISTER Slave 2

MODBUS Address	IEC61131-3 SYNTAX
0	%MW10128.0
1	%MW10130.0
2	%MW10132.0
3	%MW10134.0
4	%MW10136.0
5	%MW10138.0
6	%MW10140.0
7	%MW10142.0
8	%MW10144.0
9	%MW10146.0
...	...
2048	%MW14224.0
2049	%MW14226.0
2050	%MW14228.0
2051	%MW14230.0
2052	%MW14232.0
...	...
4088	%MW18304.0
4089	%MW18306.0
4090	%MW18308.0
4091	%MW18310.0
4092	%MW18312.0
4093	%MW18314.0
4094	%MW18316.0
4095	%MW18318.0

Once the Modbus port has been configured, the slave agent runs in the background and automatically updates the above indicated memory areas. However, a function block is also available to check the status of the slave agent: MODBUS_SLAVE_STATUS.

3-2-1 MODBUS_SLAVE_SETTINGS

Function Block setting and activating the Modbus slave agent.

FB Prototype

MODBUS_SLAVE_SETTINGS

Input description

Label	Type	Description	Range
Channel	USINT	Agent in question (Slave 1 or Slave2)	1, 2
Node_id	USINT	Node_Id of Modbus node on bus	1...247
Timeout	UINT	Timeout in units of 100 ms*	0...65535

Note: ***** if timeout = 0 the timeout computation function is disabled

Output description

Label	Type	Description	Range
Oprs	USINT	Execution Result	**0:** Ok, **1:** Port not available, **2:** Invalid Node_id

Description

This FB sets the node_id and timeout parameters of the Modbus slave agent. After execution of the function block, if successful, the Modbus Slave agent is already operative and responds to any message it receives. The timeout parameter, as explained above, communicates the maximum interval between two consecutive queries to the agent. If this interval is exceeded, the agent does not modify its operation, but simply indicates that the timeout has expired the next time the Function Block MODBUS_SLAVE_STATUS is called. To evaluate the timeout, the counter starts at the end of the execution of current Function Block and is reset every time a new query is received.

3-2-2 MODBUS_SLAVE_STATUS

Function Block checking the status of the Modbus agent.

FB Prototype

MODBUS_SLAVE_STATUS

Input description

Label	Type	Description	Range
Channel	USINT	Agent in question (Slave 1 or Slave2)	1,2

Output description

Label	Type	Description	Range
Oprs	USINT	Function Block Execution Result	0, 1, 4, 8 (see description)
Status	USINT	Communication Status	0, 1, 2, 3 (see description)

Description

Each time the Function Block is executed, all parameters which store the agent status are reset so that the result returned by the Function Block can be observed as an indication of what has occurred in the period between the last and the current calls.

The comparative status and oprs analysis can be used to assess the overall status of the agent. The status parameter can acquire the following values associated to the following meanings.

Value	Meaning
0	No operation / OK
1	At least one memory value updated
2	Timeout expired
3	Agent error event

The Oprs parameter can acquire the following values:

Value	Meaning
0	No operation / OK
1	Port not available
2	Timeout expired
4	CRC error
8	Exception

The conditions coded by the status parameter are not mutually exclusive, i.e. a condition may occur in which a timeout has expired but a new memory value has also been updated. For this reason, the diverse conditions have been coded according to increasing priority. The condition with value 3 has the highest priority; therefore, if such condition occurs, it will be surely indicated while possible lower priority conditions will not.

3-3 Data access function for Modbus slaves

Five function blocks are available which grant access to the memory areas dedicated to the Modbus slaves differently than using percentage variables. Writing and reading whole register and bit blocks are much easier and particularly:

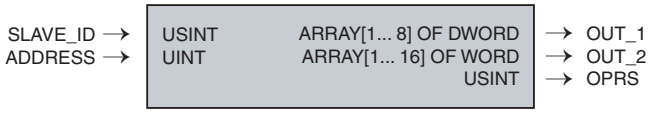
- reading in the register-dedicated memory area, starting from a definite Modbus address, both 16-register blocks, available in a 16-element vector, and 4-byte blocks, available in an 8-element vector.
- Writing the value of a single register and indicating the desired Modbus address.
- Writing 4 data bytes starting from a desired Modbus address.
- Reading 16-bit groups starting from a definite address of the memory area dedicated to the digital I/O.
- Writing a single bit and indicating the corresponding Modbus address.

3-3-1 MODBUS_GET_SLAVE_DATA

Function Block for data group reading the memory area dedicated to slave registers.

FB Prototype

MODBUS_GET_SLAVE_DATA



Input description

Label	Type	Description	Range
SLAVE_ID	USINT	Agent in question (slave 1 or slave 2)	1, 2
ADDRESS	UINT	Starting address	1, 4081

Output description

Label	Type	Description	Range
OUT_1	ARRAY[1... 8] OF DWORD	16 register vector in 4 bytes	
OUT_2	ARRAY[1... 16] OF WORD	16 register vector	
OPRS	USINT	Execution result	0, 1, 2, 3, 4

Description

Function Block enabling, given a definite initial memory address of register zone, two output vectors holding:

- the values of 16 consecutive registers starting from the initial address. The vector holds therefore 16 elements with 2 bytes each.
- the values of 16 consecutive registers starting from the initial address in 4-byte groups. The vector holds therefore 8 elements with 4 bytes each.

As the maximum register number is 4096 for each slave, the allowed address interval ranges from 1 to 4081.

The OPRS parameter can acquire the following values:

Value	Meaning
0	Block execution terminated successfully
1	Illegal address
3	Invalid protocol (different from 1 or 2)

The error condition with value 1 has the highest priority and if the condition occurs other possible errors are not diagnosed.

3-3-2 MODBUS_SET_WORD_DATA

Function Block enabling to write a register.

FB Prototype

MODBUS_SET_WORD_DATA

Input description

Label	Type	Description	Range
SLAVE_ID	USINT	Agent in question (slave 1 or slave 2)	1, 2
ADDRESS	UINT	Register address to be written	1, 4096
DATA	WORD	Value to be written in the register	

Output description

Label	Type	Description	Range
OPRS	USINT	Execution result	0, 1, 2, 3, 4

Description

The Function Block enables to write a value in the register specified by the input address. The block enables to write directly in the memory areas dedicated to the Modbus slaves without percentage variable declaration. The Function Block has simply to be called within a cycle and data and addresses to be entered to write on several registers.

The OPRS parameter can acquire the following values:

Value	Meaning
0	Block execution terminated successfully
1	Illegal address
3	Invalid protocol (different from 1 or 2)

The error condition with value 1 has the highest priority and if the condition occurs other possible errors are not diagnosed.

3-3-3 MODBUS_SET_DWORD_DATA

Function Block enabling to write in two contiguous registers (4 bytes).

FB Prototype

MODBUS_SET_DWORD_DATA

Input description

Label	Type	Description	Range
SLAVE_ID	USINT	Agent in question (slave 1 or slave 2)	1, 2
ADDRESS	UINT	Starting address	1, 4095
DATA	DWORD	Value to be written in two contiguous registers	

Output description

Label	Type	Description	Range
OPRS	USINT	Execution result	0, 1, 2, 3, 4

Description

The Function Block writes a 32-bit value in the registers, using two locations with two bytes each. Starting from the specified input address the 4 contiguous bytes are used to hold the value. A 32-bit value can be written in the last 4 bytes of the memory area dedicated to the registers if the maximum value of the allowed input address is 4095.

The OPRS parameter can acquire the following values:

Value	Meaning
0	Block execution terminated successfully
1	Illegal address
3	Invalid protocol (different from 1 or 2)

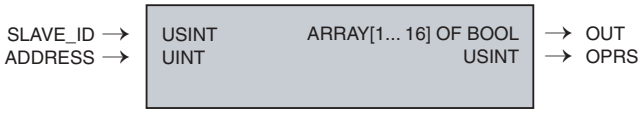
The error condition with value 1 has the highest priority and if the condition occurs other possible errors are not diagnosed.

3-3-4 MODBUS_GET_DIGITAL_SLAVE

Function Block for 16 digital value blocks reading.

FB Prototype

MODBUS_GET_DIGITAL_SLAVE



Input description

Label	Type	Description	Range
SLAVE_ID	USINT	Agent in question (slave 1 or slave 2)	1, 2
ADDRESS	UINT	Starting address	1, 128

Output description

Label	Type	Description	Range
OUT	ARRAY[1... 16] OF BOOL	Vector holding the 16 readings	TRUE/FALSE
OPRS	USINT	Execution result	0, 1, 2, 3, 4

Description

The Function Block reads the 16-bit blocks in the memory locations dedicated to the digital I/O. If a block exceeding the I/O dedicated size is read, the bits beyond such limit are set to zero. Starting from the initial address, the digital values are read and made available in a 16-element vector.

The OPRS parameter can acquire the following values:

Value	Meaning
0	Block execution terminated successfully
1	Illegal address
3	Invalid protocol (different from 1 or 2)

The error condition with value 1 has the highest priority and if the condition occurs other possible errors are not diagnosed.

3-3-5 MODBUS_SET_DIGITAL_SLAVE

Function Block for digital I/O writing.

FB Prototype

MODBUS_SET_DIGITAL_SLAVE

Input description

Label	Type	Description	Range
SLAVE_ID	USINT	Agent in question (slave 1 or slave 2)	1, 2
ADDRESS	UINT	I/O address on which to write	1, 128
DATA	BOOL	Bit value being set	TRUE, FALSE

Output description

Label	Type	Description	Range
OPRS	USINT	Execution result	0, 1, 2, 3, 4

Description

The Function Block writes individually the memory locations dedicated to the digital I/O. According to the input specified memory address, the memory locations dedicated to the I/O can be written without declaring the corresponding percentage variables. Several digital I/Os can be written by specifying the Function Block within a cycle.

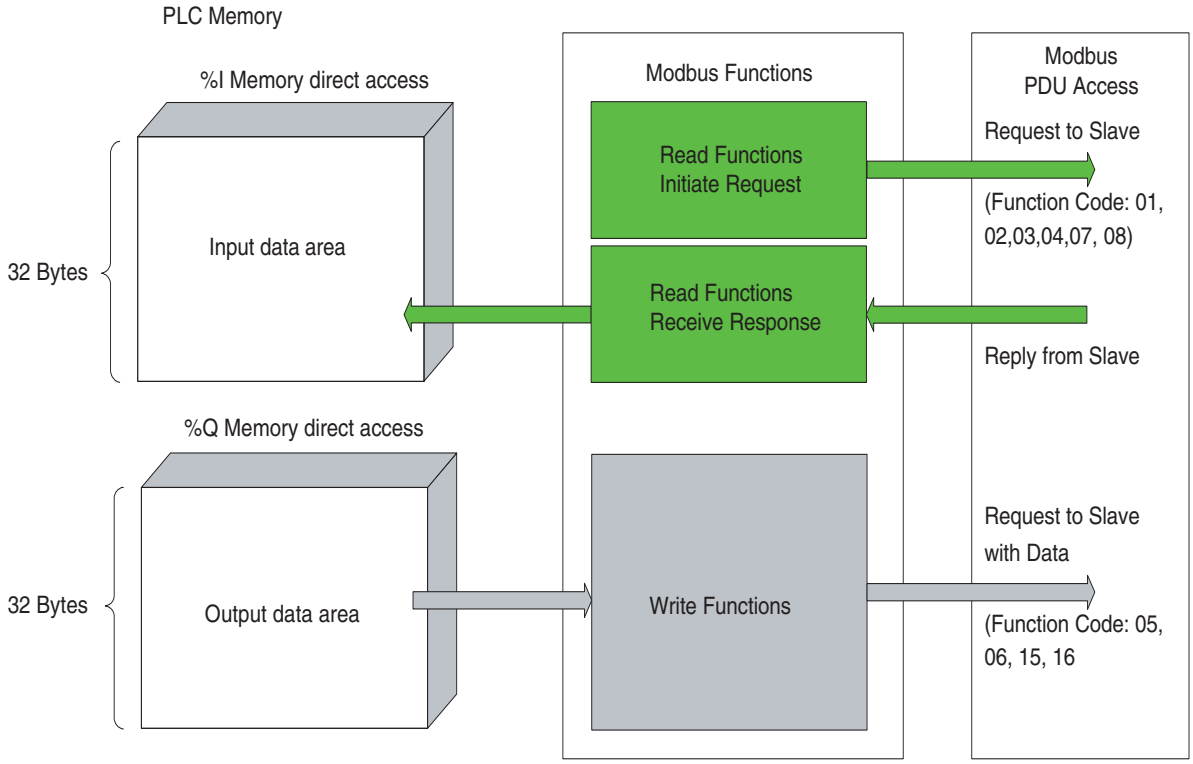
The OPRS parameter can acquire the following values:

Value	Meaning
0	Block execution terminated successfully
1	Illegal address
3	Invalid protocol (different from 1 or 2)

The error condition with value 1 has the highest priority and if the condition occurs other possible errors are not diagnosed.

3-4 Master modbus protocol

It is possible to initiate a protocol which provides the fundamental functions of a modbus Master on one of the serial ports of the Ascon CPUs. Fundamental functions constitute the possibility to generate a serial line query compliant with modbus specifications. The implementation of the Master defines a memory area, accessible via the IEC61131-3 programming environment, which stores the data that the Modbus Master agent must send to another memory area where the Master agent places the data received in response to its queries. The following figure illustrates the concept on which this modbus Master protocol implementation is based.



The supported functions are shown in the following figure.

				Function Codes		
				Code	Sub code	(hex)
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	02		02
		Internal Bits Or Physical Coils	**Read Coils**	01		01
			Write Single Coil	05		05
			Write Multiple Coils	15		0F
	16 bit access	Physical Input Registers	**Read Input Register**	04		04
		Internal Registers Or Physical Output Registers	**Read Holding Registers**	03		03
			Write Single Register	06		06
			Write Multiple Registers	16		10
Diagnostics			**Read Exception status**	07		07
			Diagnostics	08	0000	08

The two memory areas designated to input and output data storage are shown in the table below:

Protocol Agent	Protocol Memory Areas	PLC Memory Mapping
ModBus Master	Input Data	%I1000.0 to %I1031.7
	Output Data	%Q1000.0 to %Q1031.7

The IEC61131-3 programmer features two Function Blocks:

- MODBUS_MASTER_EXECUTE
- MODBUS_MASTER_STATUS

They control communications with the slaves connected to the modbus network.

One important consideration must be borne in mind when using the modbus master agent. When calling MODBUS_MASTER_EXECUTE function, one of the supported modbus functions must be assigned as a parameter. Thus, the agent interprets the byte sequence to be read from the Input Data memory area or to be placed in the Output Data Memory Area. If a function is assigned which processes registers, the two input and output memory areas are interpreted as follows:

Description	Input Area Syntax	Output Area Syntax
First register involved in communication	%IW1000.0	%QW1000.0
Second register involved in communication	%IW1002.0	%QW1002.0
Third register involved in communication	%IW1004.0	%QW1004.0
Fourth register involved in communication	%IW1006.0	%QW1006.0

If instead a function is assigned which processes individual bits, the two input and output memory areas are interpreted as follows:

Description	Input Area Syntax	Output Area Syntax
First bit involved in communication	%IX1000.0	%QX1000.0
Second bit involved in communication	%IX1000.1	%QX1000.1
Third bit involved in communication	%IX1000.2	%QX1000.2
Fourth bit involved in communication	%IX1000.3	%QX1000.3

The programmer is completely responsible for properly filling the output data area before sending a message and reading the input data area after the transaction is completed. The programmer can use the function block MODBUS_MASTER_STATUS to monitor the progress status of the queries.

3-4-1 MODBUS_MASTER_EXECUTE

Function Block executing a query in compliance with the modbus protocol.

FB Prototype

MODBUS_MASTER_EXECUTE

Input description

Label	Type	Description	Range
Slave_id	USINT	Slave address to which the query is sent	0...247
Funct	USINT	Query function	
Addr	UINT	Register and slave bit addresses being worked with	
N_of_elements	UINT	Number of registers or bits being queried	
Tiemout	UINT	Timeout in units of 100 ms*	0...65535
Rest	BOOL	Blocks current query	0...1

Note: ***** if timeout = 0 the timeout computation function is disabled

Output description

Label	Type	Description	Range
Oprs	USINT	Function Block Execution Result	**0:** Ok; **1:** port not available, **2:** agent busy, **4:** invalid address or function

Description

This function block starts the query response process characteristic of modbus communication. The function block defines the addressee of the query, the function to be executed, the address and the number of registers or bits to be queried. This Function Block assumes that the data to be sent reside in the output data memory area, while the input memory area contains the requested data. The modbus master agent checks the time from the moment the query is initiated and compares it to the timeout set at the time the Function Block is called. The query and response process is aborted if the timeout expires. The query and response process can also be intentionally aborted by calling the Function Block with the Rest input set to TRUE.

3-4-2 MODBUS_MASTER_STATUS

Function Block using the Latch and Time functions of the DI-16LV module.

FB Prototype

MODBUS_MASTER_STATUS

Output description

Label	Type	Description	Range
Datarec	USINT	Number of bytes received	
Excode	USINR	Modbus protocol error code	
Oprs	USINT	Function Block Execution Result	0,1,4,8,16 (see description)
Status	USINT	Communication Status	0,1,2,3,4 (see description)

Description

Each time the Function Block is executed, all parameters which store the agent status are reset so that the result returned by the Function Block can be observed as an indication of what has occurred in the period between the last and the current calls.
The comparative status and oprs analysis can be used to assess the overall status of the agent. The **Status** parameter can acquire the following values associated to the following meanings.

Value	Meaning
0	No operation / OK
1	Transaction in progress
2	Available data
3	Timeout expired
4	Agent error event

The **Oprs** parameter can acquire the following values:

Value	Meaning
0	No operation / OK
1	Port not available
2	Timeout expired
4	CRC error
8	Exception
16	Error: see ExCode

The conditions coded by the status parameter are not mutually exclusive, i.e. a condition may occur in which a timeout has expired but a new memory value has also been updated. For this reason, the diverse conditions have been coded according to increasing priority. The condition with value 4 has the highest priority; therefore, if such condition occurs, it will be surely indicated while possible lower priority conditions will not.

3-4-3 Application example of the modbus communication port

This section describes the code of a simple application which controls two serial ports, one of which is configured as a master and the other as a slave. The program is divided into 3 tasks: one to initialise the ports, one to execute the master activities and one to execute the slave activities. The slave task must cyclically assign 5 different values to 5 registers, while the master task must acquire the content of the five registers via modbus protocol and compare them with the values they should have according to the slave operations. If the values read in the registers and those assigned by the slave are the same, the master increments a counter and goes to the next step.

3-4-4 Global application variables

The program's tasks share a number of global variables which are code-declared as follows:

```
VAR_GLOBAL

baud : USINT;
parity : USINT;
data_length : USINT;
stop_bit : USINT;
init_port: BOOL := true;
error: BOOL;
node_id : USINT;

next_step: BOOL := true;
sequence_read: ARRAY [1..5] OF WORD;
sequence_write: ARRAY [1..5] OF WORD;

END_VAR
```

In addition to serial port configuration parameters, a number of parameters such as **init_por**, **error** and **next_step** cyclically control the execution of the code of the running tasks depending on the status of the other tasks. The two arrays, **sequence_read** and **sequence_write**, contain the values written in the registers and the values read via the modbus.

3-4-5 HW port initialisation

This task initialises the serial ports and assigns them the desired protocols. The task code variable declaration and executable code are shown below:

Variable declaration

```
VAR_EXTERNAL

baud : USINT;
parity : USINT;
data_length: USINT;
stop_bit : USINT;
init_port: BOOL;
error: BOOL;
node_id : USINT;

END_VAR

VAR_GLOBAL

END_VAR

VAR

timeout: UINT;
oprs: USINT;
open_serial_com: OPEN_SERIAL_COMM;
modbus_slave_setting: MODBUS_SLAVE_SETTINGS;
close_serial_com: CLOSE_SERIAL_COMM;

END_VAR
```

Task code

```
if init_port then

baud := 3;
parity := 0;
data_length:= 8;
stop_bit:= 1;
init_port:= false;
node_id := 16#11;
timeout:= 10;
error:= FALSE;

close_serial_com(
PORT := 0);

close_serial_com(
PORT := 1);

open_serial_com(
PORT      := 0,
BAUD      := baud,
PARITY    := parity,
DATA      := data_length,
STOP      := stop_bit,
PROTOCOL  := 1
|
oprs      := OPRS);

if oprs > 0 then
error     := true;
end_if;

open_serial_com(
PORT      := 1,
BAUD      := baud,
PARITY    := parity,
DATA      := data_length,
STOP      := stop_bit,
PROTOCOL  := 3
|
oprs      := OPRS);

if oprs > 0 then
error     := true;
end_if;

modbus_slave_setting(
CHANNEL   := 1,
NODE_ID   := node_id,
TIMEOUT   := timeout
|
oprs      := OPRS);

if oprs > 0 then
error     := true;
end_if;
end_if;
```

It must be noted how before opening the serial ports, the Function Block is called to close them. From a logical point of view this might seem incorrect, but the open functions are designed to return an error if the port is already busy. If during the development stage the ports are not closed before downloading a new version of the firmware, they remain busy, and the system will not allow them to be used when the program is launched.

3-4-6 Master code

This task controls modbus master communication.

Variable declaration

```
VAR_EXTERNAL

baud : USINT;
parity : USINT;
data_length : USINT;
stop_bit : USINT;
init_port: BOOL;
error: BOOL;
node_id : USINT;

next_step: BOOL;
sequence_read: ARRAY [1..5] OF WORD;
sequence_write: ARRAY [1..5] OF WORD;

END_VAR

VAR_GLOBAL

END_VAR

VAR
status    : USINT;
datarec   : USINT;
excode    : USINT;
timeout_err: UDINT := 0;
err_err   : UDINT := 0;
mb_exec_busy_err: UDINT := 0;
mb_exec_no_port: UDINT := 0;
mb_exec_func_err: UDINT := 0;
counter_ok: UDINT := 0;
counter_ko: UDINT := 0;

modbus_master_exec: MODBUS_MASTER_EXECUTE;
modbus_master_statuss: MODBUS_MASTER_STATUS;

oprs: USINT;

at %IW1000.0:word;
at %IW1002.0:word;
at %IW1004.0:word;
at %IW1006.0:word;
at %IW1008.0:word;
END_VAR
```

Task code

```
if init_port = false then
if error = false then
if next_step = false then
modbus_master_statuss(
                |
                datarec:= DATAREC,
                excode:= EXCODE,
                status := STATUS,
                oprs := OPRS);

case status of
     0: (* next activity associated to the master*)
          modbus_master_exec(
                SLAVE_ID  := node_id,
                FUNCT     := 4,
                ADDR      := 1,
                N_OF_ELEMENTS  := 5,
                TIMEOUT        := 10,
                REST           := FALSE
                |
                oprs := OPRS);
          case oprs of
```

```
                    1: mb_exec_no_port := mb_exec_no_port + 1;
                    2: mb_exec_busy_err := mb_exec_busy_err + 1;
                    4: mb_exec_func_err := mb_exec_func_err + 1;
               end_case;
          1: (* the master is processing the last function required*);
          2: (* the master has ended the last activity required*);
               sequence_read[1] := %IW1000.0;
               sequence_read[2] := %IW1002.0;
               sequence_read[3] := %IW1004.0;
               sequence_read[4] := %IW1006.0;
               sequence_read[5] := %IW1008.0;

               if sequence_read[1] <> sequence_write[1] then
               counter_ko := counter_ko + 1;
               elsifsequence_read[2] <> sequence_write[2] then
                    counter_ko := counter_ko + 1;
               elsifsequence_read[3] <> sequence_write[3] then
                    counter_ko := counter_ko + 1;
               elsifsequence_read[4] <> sequence_write[4] then
                    counter_ko := counter_ko + 1;
               elsifsequence_read[5] <> sequence_write[5] then
                    counter_ko := counter_ko + 1;
               else
                    counter_ok := counter_ok + 1;
               end_if;

               next_step := true;

          3: (* the timeout has expired during the last activity required*)
               timeout_err := timeout_err + 1;
          4: (* an error occurred during the last activity required*)
               err_err := err_err + 1;
end_case;
end_if;
end_if;
end_if;
```

3-4-7 Slave code

This task works with the registers which are addressed by the slave agent. The code does not refer to any Function Block controlling the agent since the agent runs in the background and makes a certain memory area available, while the IEC1131-3 program only deals with variables in the memory area shared with the agent. It is however possible – not shown in this example – to obtain information from the modbus slave agent via the Function Block MODBUS_SLAVE_STATUS.

Variable declaration

```
VAR_EXTERNAL

baud : USINT;
parity : USINT;
data_length : USINT;
stop_bit : USINT;
init_port: BOOL;
error: BOOL;
node_id : USINT;

next_step: BOOL;
sequence_read: ARRAY [1..5] OF WORD;
sequence_write: ARRAY [1..5] OF WORD;

END_VAR

VAR_GLOBAL

END_VAR

VAR
sequence:USINT := 1;
```

```
at %MW1128.0:word;
at %MW1130.0:word;
at %MW1132.0:word;
at %MW1134.0:word;
at %MW1136.0:word;
END_VAR
```

Task code

```
if init_port = false then
if error = false then
if next_step = true then

case sequence of
     1:   sequence_write[1] := 16#0102;
          sequence_write[2] := 16#0304;
          sequence_write[3] := 16#0506;
          sequence_write[4] := 16#0708;
          sequence_write[5] := 16#090A;

     2:   sequence_write[1] := 16#0304;
          sequence_write[2] := 16#0506;
          sequence_write[3] := 16#0708;
          sequence_write[4] := 16#090A;
          sequence_write[5] := 16#0102;

     3:   sequence_write[1] := 16#0506;
          sequence_write[2] := 16#0708;
          sequence_write[3] := 16#090A;
          sequence_write[4] := 16#0102;
          sequence_write[5] := 16#0304;

     4:   sequence_write[1] := 16#0708;
          sequence_write[2] := 16#090A;
          sequence_write[3] := 16#0102;
          sequence_write[4] := 16#0304;
          sequence_write[5] := 16#0506;

     5:   sequence_write[1] := 16#090A;
          sequence_write[2] := 16#0102;
          sequence_write[3] := 16#0304;
          sequence_write[4] := 16#0506;
          sequence_write[5] := 16#0708;
end_case;

%MW1128.0 := sequence_write[1];
%MW1130.0 := sequence_write[2];
%MW1132.0 := sequence_write[3];
%MW1134.0 := sequence_write[4];
%MW1138.0 := sequence_write[5];

if sequence = 5 then
     sequence := 1;
else
     sequence := sequence + 1;
end_if;

next_step := false;
end_if;
end_if;
end_if;
```

3-5 ASCII serial

The following Function Blocks have been created to swap ASCII strings on a serial port via CPU without implementing a specific protocol.

3-5-1 CONV_ASCII_TO_CHAR

FB Prototype

CONV_ASCII_TO_CHAR

Input parameters

Label	Type	Description
ASCII	BYTE	Binary code of the input string character

Output parameters

Label	Type	Description
CHAR	STRING(1)	Alphanumeric string character has to be converted in the equivalent binary ASCII code

Description

This FB can be used to convert a binary ASCII code in the equivalent alphanumerical character.

3-5-2 CONV_CHAR_TO_ASCII

FB Prototype

CONV_CHAR_TO_ASCII

Input parameters

Label	Type	Description
CHAR	STRING(1)	Alphanumeric string character has to be converted in the equivalent binary ASCII code

Output parameters

Label	Type	Description
ASCII	BYTE	Binary code of the input string character

Description

This FB can be used to convert any alphanumerical character in the equivalent binary ASCII code.

3-5-3 SERIAL_IO_CONFIG

FB Prototype

SERIAL_IO_CONFIG

Input parameters

Label	Type	Description
PORT	USINT	Port on which the serial agent is initiated. Admissible values: µPAC: 0 (X0) or 1 (X1), SigmaPAC: 0 (X1), 1 (X3) or 2 (X4)
SERVICE	USINT	Requested service

Output parameters

Label	Type	Description
N_OF_BYTE	USINT	Number of bytes related to the requested service
OPRS	USINT	Service status

Description

This FB can be used to request some functions to the serial ASCII agent. The input parameter “SERVICE” the type of function desired. The available functions are:

Value	Description
0	Number of bytes present in the input buffer
1	Number of bytes present in the output buffer
2	Send immediately all the bytes present in the output buffer and leave it empty
3	Reset immediately all the output buffer bytes
4	Reset immediately all the input buffer bytes

The FB will returns the status of the operated service on the output “OPRS” parameter. The returned values are:

Value	Description
0	Function executed successfully
1	Port not configured for the ASCII serial protocol
2	Peripheral driver problems
3	Unavailable number of requested bytes in the input buffer
4	Number of bytes present in the input buffer less then requested

3-5-4 SERIAL_IO_READ

FB Prototype

SERIAL_IO_READ

Input parameters

Label	Type	Description
PORT	USINT	Port on which the serial agent is initiated: admissible values: µPAC: 0 (X0) or 1 (X1), SigmaPAC: 0 (X1), 1 (X3) or 2 (X4)
N_OF_BYTE	USINT	Desired amount of bytes to read

Output parameters

Label	Type	Description
BUFFER	STRING(100)	String containing the requested bytes
OPRS	USINT	Operation status

Description

This FB can be used to receive a desired amount of bytes within the serial ASCII agent. ONLY when the specified amount of bytes will be received they will be loaded in the output BUFFER. If the amount of bytes differs from the requested quantity, the FB will return an error. The values returned by the “OPRS” parameter are:

Value	Description
0	Operation executed successfully
1	Port not configured for the ASCII serial protocol
2	Peripheral driver problems
3	Unavailable number of requested bytes in the input buffer
4	Number of bytes present in the input buffer less then requested

3-5-5 SERIAL_IO_READ_BYTE

FB Prototype

SERIAL_IO_READ_BYTE

Input parameters

Label	Type	Description
PORT	USINT	Port on which the serial agent is initiated: admissible values: µPAC: 0 (X0) or 1 (X1), SigmaPAC: 0 (X1), 1 (X3) or 2 (X4)
N_OF_BYTE	USINT	Desired amount of bytes to read

Output parameters

Output	Type	Description
BUFFER	ARRAY[1..100] of BYTE	String containing the requested bytes
OPRS	USINT	Operation status

Description

This FB can be used to receive a desired amount of bytes within the serial ASCII agent. ONLY when the specified amount of bytes will be received they will be loaded in the output BUFFER. If the amount of bytes differs from the requested quantity, the FB will return an error. The values returned by the “OPRS” parameter are:

Value	Description
0	Operation executed successfully
1	Port not configured for the ASCII serial protocol
2	Peripheral driver problems
3	Unavailable number of requested bytes in the input buffer
4	Number of bytes present in the input buffer less then requested

3-5-6 SERIAL_IO_WRITE

FB Prototype

SERIAL_IO_WRITE

Input parameters

Label	Type	Description
PORT	USINT	Port on which the serial agent is initiated: admissible values: µPAC: 0 (X0) or 1 (X1), SigmaPAC: 0 (X1), 1 (X3) or 2 (X4)
N_OF_BYTE	USINT	Desired amount of bytes to write
BUFFER	STRING (100)	String containing the bytes to be sent

Output parameters

Output	Type	Description
OPRS	USINT	Operation status

Description

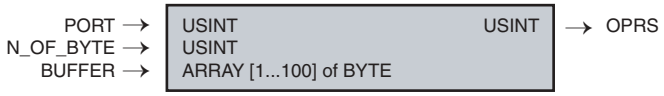
This FB can be used to send a desired amount of bytes within the serial ASCII agent. The values returned by the “OPRS” parameter are:

Value	Description
0	Operation executed successfully
1	Port not configured for the ASCII serial protocol
2	Attempt to write a number of bytes that exceeds the allowed limit

3-5-7 SERIAL_IO_WRITE_BYTE

FB Prototype

SERIAL_IO_WRITE_BYTE



Input parameters

Label	Type	Description
PORT	USINT	Port on which the serial agent is initiated: admissible values: µPAC: 0 (X0) or 1 (X1), SigmaPAC: 0 (X1), 1 (X3) or 2 (X4)
N_OF_BYTE	USINT	Desired amount of bytes to write
BUFFER	ARRAY [1... 100] of BYTE	Array of bytes which contain the bytes to be sent

Output parameters

Label	Type	Description
OPRS	USINT	Operation status

Description

This FB can be used to send a desired amount of bytes within the serial ASCII agent. The values returned by the “OPRS” parameter are:

Value	Description
0	Operation executed successfully
1	Port not configured for the ASCII serial protocol
2	Attempt to write a number of bytes that exceeds the allowed limit

Chapter 4
CanOPEN (for CU-02 unit only)

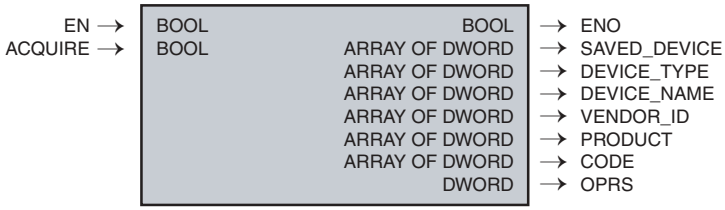
4-1 Canopen FBs

4-1-1 CANOPEN_AUTOLEARN

It performs the network scan and stores the main important parameters of the connected I/O modules.

FB Prototype

CANOPEN_AUTOLEARN



Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
ACQUIRE	BOOL	Start the command execution	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
SAVED_DEVICE	ARRAY OF BOOL	Array of the present addresses	TRUE/FALSE
DEVICE_TYPE	ARRAY OF DWORD	Read device types array	
DEVICE_NAME	ARRAY OF DWORD	Device names array	
VENDOR_ID	ARRAY OF DWORD	Vendor IDs array	
PRODUCT CODE	ARRAY OF DWORD	Product codes array	
OPRS	DWORD	OPeration result	**0** Operation successful **1** Memory management problems **2** Network problems

Description

Setting the input ACQUIRE to TRUE it is possible to run the command to capture and save the network image. Recalling the function block with the ACQUIRE value set to FALSE erases from the memory a memory dump prebviously saved.

When you want to scan the image of the current network, it is not necessary to erase the memory image currently stored.
The 5 return arrays of this function block contain the identity information of the modules found on the network during the autolearn.

All the array elements correspond to the module with address equal to the index, if an identity field is not implemented it will be filled with ZERO. The SAVED_DEVICE array, in particular when has value TRUE, indicates that at a particular address, there is a module connected or if it has value FALSE, that no modules have been detected.

4-1-2 CANOPEN_DEVICE_COMPARE

It compares the characteristics of an I/O module already recognized and saved by the system, between the two network images (flash and RAM memory).

FB Prototype

CANOPEN_DEVICE_COMPARE

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
DEVICE	BYTE	Address of the device to be analyzed	

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
STATUS	WORD	Result of the module comparison	
OPRS	WORD	Function block execution result	

Description

The device status contains a word with the information of the comparison between the analized module and the image saved in flash memory. The information is coded using the bits as explained in the table that follows:

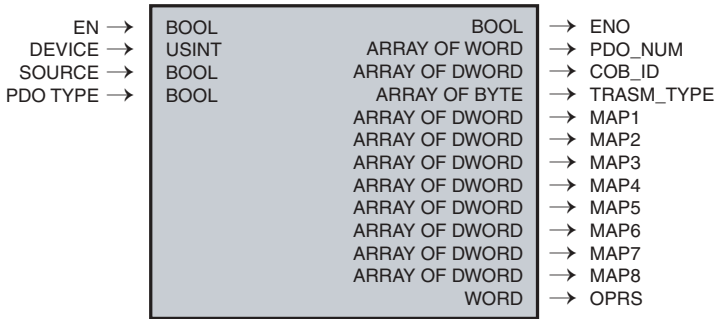
Information	Code
Module not present and not included in the image memory	0x0000
Module present and correspondent to the image memory	0x0001
Module present, but different for mapping	0x0002
Module present but different for identity	0x0004
Module present, but not expected by the image in memory	0x0008
Module not present, but expected by the image in memory	0x0010

4-1-3 CANOPEN_GET_DEVICE_COM_PAR

If used in conjunction with "CanOPEN_Network_Compare" and "CanOPEN_Autolearn", it returns the I/O module mapping information stored by the previously by the system. It should be used after the network scan has been saved into the memory.

FB Prototype

CANOPEN_GET_DEVICE_COM_PAR


EN → BOOL
DEVICE → USINT
SOURCE → BOOL
PDO TYPE → BOOL
BOOL → ENO
ARRAY OF WORD → PDO_NUM
ARRAY OF DWORD → COB_ID
ARRAY OF BYTE → TRASM_TYPE
ARRAY OF DWORD → MAP1
ARRAY OF DWORD → MAP2
ARRAY OF DWORD → MAP3
ARRAY OF DWORD → MAP4
ARRAY OF DWORD → MAP5
ARRAY OF DWORD → MAP6
ARRAY OF DWORD → MAP7
ARRAY OF DWORD → MAP8
WORD → OPRS

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
DEVICE	USINT	Enquired Device	
SOURCE	BOOL	Source of the information requested	TRUE = Autolearned Network FALSE = Compared Network
PDO_TYPE	BOOL	Defines which part of the communication information necessary to pass	TRUE = TPDO FALSE = RPDO
EN	BOOL	Enable In	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
PDO_NUM	ARRAY OF WORD	PDO index (0xFFFF no PDO)	
COB_ID	ARRAY OF DWORD	COB_ID of the mapped PDOs	
TRASM_ TYPE	ARRAY OF BYTE	Trasmission Type of the mapped PDOs	
MAP1	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP2	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP3	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP4	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP5	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP6	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP7	ARRAY OF DWORD	Mapping of the objects in PDOs	
MAP8	ARRAY OF DWORD	Mapping of the objects in PDOs	
OPRS	WORD	Function block execution result	

Each of the output arrays is composed of 4 elements

Description

The field OPRS must be interpreted according to the following table:

Description	Code
Recovered data valid	0x0000
Device address not valid	0x0001
Requested source not available	0x0002

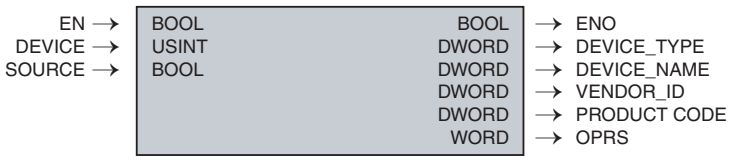
4-1-4 CANOPEN_GET_DEVICE_ID

Using this Function Block is possble to obtain the identity information of a module in the same form saved by the system; to obtain these information the system uses the Function Blocks "CanOPEN_Network_Compare" and "CanOPEN_Autolearn".

I.e.: you can obtain identity data from a module from the network image stored in memory, or the data of the module which has been scanned at the last compare.

FB Prototype

CANOPEN_GET_DEVICE_ID



Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
DEVICE	USINT	Device of interest	
SOURCE	BOOL	Source of the requested information	TRUE = Freezed Network, FALSE = Compared Network

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
DEVICE_TYPE	DWORD	Array of the acquired device type	
DEVICE_NAME	DWORD	Array of the device names	
VENDOR_ID	DWORD	Array of the vendor IDs	
PRODUCT CODE	DWORD	Array of the product codes	
OPRS	WORD	Execution results	

OPRS field must be interpreted according to the following table:

Information	Code
Acquired data are valid	0x0000
Device address not valid	0x0001
The requested source is not available	0x0002

4-1-5 CANOPEN_NETWORK_COMPARE

It performs a comparison between the connected I/O modules network image and the one already saved into the system memory, with regards to the previously described parameters.

FB Prototype

CANOPEN_NETWORK_COMPARE

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
DEVICE_STATUS	ARRAY OF WORD	Comparison result for each module	
NETWORK_STATUS	WORD	Comparison result for the entire network	

Description The device status contains an array of word index that contains, for each, information about the modules found at the corresponding addresses. The information is encoded in bits in the table below:

Information	Code
Module not present and not expected from the image in memory	0x0000
Present and corresponding to the image module in memory	0x0001
Present form, but for Different Mapping	0x0002
Present form, but for Different Identities	0x0004
Module present, but not provided by the image in memory	0x0008
Module not present, but expected from the image in memory	0x0010

Network_status for the legend of the output status is as follows:

Information	Code
Network identical to the image in memory	0x0000
Network identical to the presence of all modules with the expected waiting identity but different mapping	0x0001
Network identical to the presence of all forms, but some elements do not match for identity	0x0002
Network mismatch	0x0004
No network in memory	0x0008
roblems during network scan	0x0010
Module not present, but expected from the image in memory	0x0010

The presence of modules that are not shown in the image of network you have in mind is ignored for the purposes of carrying identity between this network and the network of which you have a picture. To be clear if the network this has all the modules with the same identity and the same mapping saved more than other models, function block returns the current network is identical to that stored in memory.

4-1-6 CANOPEN_NETWORK_STATUS

The following block can receive information about the network status in relation to a series of events which might have involved going to change its topology, configuration, or proper operation.

FB Prototype

CANOPEN_NETWORK_STATUS

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
DEVICE STATUS	ARRAY OF BYTE	Status for each module	
NETWORK_STATUS	BYTE	Status of the entire network	

Description The status device contains an array of bytes for each index contains information on events involving the forms corresponding to the addresses. The audit event starts from the call CanOPEN_Net_Control_Start. The information is encoded in bits in the table below:

Information	Code
No significant event	0x0000
Error in the diagnostic protocol	0x0001
The module has sent a message to boot-up	0x0002
The module has changed its been in the automation of communication	0x0004
The module has sent an emergency message	0x0008

Network_status for the legend of the output status is as follows:

Information	Code
No significant event on the network	0x0000
Occurred one or more significant events	0x0001

4-1-7 CANOPEN_NET_CONTROL_START

Start the agent that executes the control of events CANopen network storing any abnormal situations.

FB Prototype

CANOPEN_NET_CONTROL_START

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
START	BOOL	Enable and disable the Agent	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE

Description This function block allows you to kick off a staff member who is running the event control CANopen network storing any abnormal situations. The initiation agent is obtained by calling the funcion block with the START parameter set to TRUE, but you can disable the function block chemotherapy agent with the START parameter set to FALSE.

4-1-8 CANOPEN_SCAN_DEVICE_COM

Through this function block can run the scan characteristics of a communication module and replace the current image of the network that the system has gained during the "CanOPEN_Network_Compare. This function block is not blocking like everyone else in this library so that it can carry out its function should be called periodically until it returns a signal logoff.

FB Prototype

CANOPEN_SCAN_DEVICE_COM

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
ENABLE	BOOL	It manages the sequence of calls for the implementation of block	TRUE/FALSE
DEVICE	BYTE	Address of the device to be analyzed	

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
CONFIRM	BOOL	End-of-execution	
ERROR	WORD	Error Code	
ERR_INFO	DWORD	Details of the errors occurred	

4-1-9 CANOPEN_SCAN_DEVICE_ID

It scans the specified device address and replace it into the network image created by the system using the "CanOPEN_Network_Compare" FB. It must be called cyclically until the end of operations.

FB Prototype

CANOPEN_SCAN_DEVICE_ID

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
ENABLE	BOOL	It manages the sequence of calls for the implementation of block	TRUE/FALSE
DEVICE	BYTE	Address of the device to be analyzed	

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
CONFIRM	BOOL	End-of-execution	
ERROR	WORD	Error Code	
ERR_INFO	DWORD	Details of the errors occurred	

4-1-10 CAN_ENABLE_CYCLIC_SYNC

Function block for enabling or locking cyclical SYNC messages.

FB Prototype

CAN_ENABLE_CYCLIC_SYNC

Input description

Label	Type	Description
SYNC_MODE	BOOL	TRUE = Enable generation of cyclical SYNC messages FALSE = Lock generation of cyclical SYNC messages
SYNC_TIME	TIME	Time between two sequential SYNC messages, or 0 for a SYNC message after each PLC cycle
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”

Description

The function block CAN_ENABLE_SYNC is used to enable or lock the generation of cyclical SYNC messages through the PLC. When enabled the control unit generates a SYNC message between two sequential PLC cycles, if the time elapsed between the last SYNC message and the new one is longer than the time specified at the SYNC_TIME input. If the time elapsed since the last SYNC is less than the SYNC_TIME, then no new SYNC message is generated.

The input value SYNC_TIME = 0 causes the control unit to conclude each PLC cycle with a SYNC message. In this case the network process image exchange occurs in synchronization with the process image exchange for the local inputs and outputs of the PLC.

At the end of the PLC cycle the control unit always tests the time requirements for sending a SYNC message. The time given at the input SYNC_TIME is therefore the minimum time between two sequential SYNC messages. The real time interval between two SYNC messages can therefore vary in a worst case by the length of an PLC cycle:

```
Tsync [worst case] = SYNC_TIME + TPLC Zyklus
```

This function block is only available on control units in "PLC with CANopen Master" mode and can only be used as alternative to the CAN_SEND_SYNC function block.

4-1-11 CAN_GET_CANOPEN_KERNEL_STATE

Function block for state query of the CANopen kernel of the local PLC.

FB Prototype

CAN_GET_CANOPEN_KERNEL_STATE

Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
STATE	WORD	State or error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”

Description

The function block CAN_GET_CANOPEN_KERNEL_STATE is used for a query about the state of the CANopen kernel of the local PLC.

4-1-12 CAN_GET_LOCAL_NODE-ID

Function block for a local node address query.

FB Prototype

CAN_GET_LOCAL_NODE-ID

Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
DEVICE	USINT	Local Node Address of the PLC

Description

The function block CAN_GET_LOCAL_NODE_ID is used for a query of the local node address of the PLC. The node address of a control unit has an influence over the availability of the various function blocks.

4-1-13 CAN_GET_STATE

Function compent for node state query of various devices.

FB Prototype

CAN_GET_STATE

Input description

Label	Type	Description
DEVICE	USINT	Address of the node to be queried
ENABLE	WORD	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
STATE	WORD	Node state corresponding to the data type “CIA405_STATE”

Description

The function block CAN_GET_STATE is used to enquire the node state for a specific device. The state query is based on monitoring by Heartbeat or Lifeguarding. Detailed information on these functions can be found in section 2.3. The return values on the output STATE have the following meaning:

UNKNOWN: The CANopen device on the given address supports neither Heartbeat nor Lifeguarding, thus the state can not be monitored. This state is also reported on an PLC without a CANopen Master if either no PLC with a CANopen Master is available in the network that supports a state transmission as described in section 2.3 or if the Master PLC in question is in a stop state (PLC program has been halted).

NOT_AVAIL: The CANopen device on the given address no longer answers Heartbeat or Lifeguarding queries and is therefore no longer available to the system.

OTHER: With the exception of the state values UNKNOWN and NOT_AVAIL the return values match the corresponding definitions of the CiA Draft Standard 301.

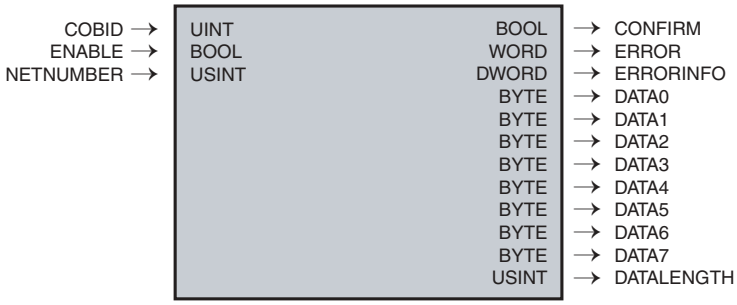
The call of the function block with DEVICE = 0 delivers the local node state of the local PLC.

4-1-14 CAN_PDO_READ8

Function block for reading a node's object entries by way of an SDO transfer.

FB Prototype

CAN_PDO_READ8



Input description

Label	Type	Description
COBID	UINT	COBID of the message to be newly entered into registration or to be erased from the registration
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to data type "CIA405_CANOPEN_KERNEL_ERROR"
ERRORINFO	DWORD	Reserved for additional error information
DATA0	BYTE	Data byte of the received CAN message
DATA1	BYTE	Data byte of the received CAN message
DATA2	BYTE	Data byte of the received CAN message
DATA3	BYTE	Data byte of the received CAN message
DATA4	BYTE	Data byte of the received CAN message
DATA5	BYTE	Data byte of the received CAN message
DATA6	BYTE	Data byte of the received CAN message
DATA7	BYTE	Data byte of the received CAN message
DATALENGTH	USINT	Length of the received CAN message

Description

The function block CAN_PDO_READ8 is used to read PDOs and CAN Layer 2 messages from the receiving buffer of the network layer. Only one access of the message registered with the help of the function block CAN_REGITER_COBID is supported. If multiple messages with the same COBID are received between two sequential calls of AN_PDO_READ8, then the most recently received message overwrites the previous one. Thus only the most current message remains stored in the receiving buffer. After a read out via the function block CAN_PDO_READ8, the corresponding message gets erased from the network layer's receiving buffer.

This prohibits multiple readings of a single message by the PLC program.

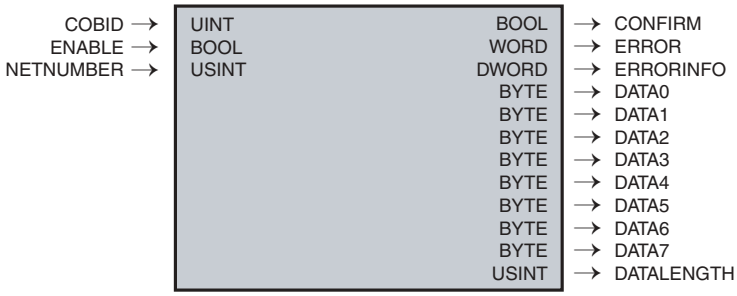
If the output CONFIRM is set to TRUE upon the return of the function block, then the elements DATA0 to DATA7 contain the individual bytes of the received message. The output ATALENGTH will then report the number of valid data bytes (beginning withDATA0). However, if the output CONFIRM is set to FALSE, then the receiving buffer does not contain any messages with the given COBID in the network layer. With CONFIRM it is possible to differentiate between a valid message of 0 bytes in length was received as opposed to no message at all.
If no message is available, then this is also shown by the error code NO_VALID_DATA_AVAILABLE on the output ERROR.

4-1-15 CAN_PDO_WRITE8

Function block for sending PDOs and CAN Layer 2 messages throught the network layer.

FB Prototype

CAN_PDO_WRITE8



Input description

Label	Type	Description
COBID	UINT	COBID of the message to be newly entered into registration or to be erased from the registration
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to data type “CIA405_CANOPEN_KERNEL_ERROR”
ERRORINFO	DWORD	Reserved for additional error information
DATA0	BYTE	Data byte of the received CAN message
DATA1	BYTE	Data byte of the received CAN message
DATA2	BYTE	Data byte of the received CAN message
DATA3	BYTE	Data byte of the received CAN message
DATA4	BYTE	Data byte of the received CAN message
DATA5	BYTE	Data byte of the received CAN message
DATA6	BYTE	Data byte of the received CAN message
DATA7	BYTE	Data byte of the received CAN message
DATALENGTH	USINT	Length of the received CAN message

Description

The function block CAN_PDO_READ8 is used to read PDOs and CAN Layer 2 messages from the receiving buffer of the network layer. Only one access of the message registered with the help of the function block CAN_REGITER_COBID is supported. If multiple messages with the same COBID are received between two sequential calls of AN_PDO_READ8, then the most recently received message overwrites the previous one. Thus only the most current message remains stored in the receiving buffer. After a read out via the function block CAN_PDO_READ8, the corresponding message gets erased from the network layer's receiving buffer.

This prohibits multiple readings of a single message by the PLC program.

If the output CONFIRM is set to TRUE upon the return of the function block, then the elements DATA0 to DATA7 contain the individual bytes of the received message. The output ATALENGTH will then report the number of valid data bytes (beginning withDATA0). However, if the output CONFIRM is set to FALSE, then the receiving buffer does not contain any messages with the given COBID in the network layer. With CONFIRM it is possible to differentiate between a valid message of 0 bytes in length was received as opposed to no message at all.
If no message is available, then this is also shown by the error code NO_VALID_DATA_AVAILABLE on the output ERROR.

4-1-16 CAN_RECV_BOOTUP

Function block for the Bootup messages of any node from the network layer's receiving buffer.

FB Prototype

CAN_RECV_BOOTUP

Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
DEVICE	USINT	Address of the node (1-127), from which a Bootup message was received
ERROR	WORD	Error code corresponding to the data type "CIA405_CANOPEN_KERNEL_ERROR"

Description

The function block CAN_RECV_BOOTUP is used for reading the Bootup messages of any node from the network layer's receiving buffer. If upon the return of the function block the output CONFIRM is set to TRUE, the output DEVICE gives the node address that the message was received from. However, if the output CONFIRM is set to FALSE, then the network layer's receiving buffer does not contain any Bootup messages.

The function block always returns the first Bootup message entered into the receiving buffer (= oldest message), the message is subsequently erased from the receiving buffer. Thus every Bootup message can only be read one time by the PLC program.

The function blocks CAN_RECV_BOOTUP_DEV and CAN_RECV_BOOTUP both access the same receiving buffer.

This function block is only available on control units in “PLC with CANopen Master” mode.

4-1-17 CAN_RECV_BOOTUP_DEV

Function block for reading Bootup messages of a specific node from the receiving buffer of the network layer.

FB Prototype

CAN_RECV_BOOTUP_DEV

Input description

Label	Type	Description
DEVICE	USINT	Address of the node (1... 127) for which the receipt of Bootup messages must be tested
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”

Description

The function block CAN_RECV_BOOTUP_DEV is used for reading the Bootup messages of a specific node from the network layer's receiving buffer. If upon the return of the function block the output CONFIRM is set to TRUE, a Bootup message was received for the reported node. However, if the output CONFIRM is set to FALSE, then the network layer's receiving buffer does not contain a Bootup message for the node in question.

After a Bootup message has been read it is erased from the receiving buffer and therefore only reported to the PLC program once. The function blocks CAN_RECV_BOOTUP_DEV and CAN_RECV_BOOTUP both access the same receiving buffer.

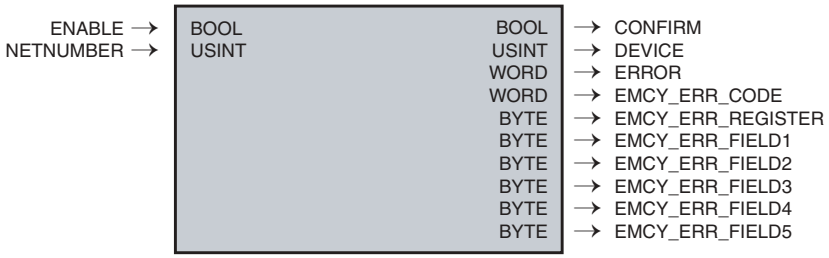
This function block is only available on control units in “PLC with CANopen Master” mode.

4-1-18 CAN_RECV_EMCY

Function block for reading emergency messages of a node from the network layer‘s receiving buffer.

FB Prototype

CAN_RECV_EMCY



Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
DEVICE	USINT	Address of the node (1-127) from which an emergency message was received
ERROR	WORD	Error code corresponding to the data type "CIA405_CANOPEN_KERNEL_ERROR"
EMCY_ERR_CODE	WORD	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_REGISTER	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD1	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD2	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD3	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD4	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD5	BYTE	Emergency error information corresponding to the CiA Draft Standard 301

Description

The function block CAN_RECV_EMCY is used for reading the emergency messages of any nodes from the network layer's receiving buffer. If upon the return of the function block the output CONFIRM is set to TRUE, the output DEVICE reports the node address from which the message was received. The elements EMCY_ERR contain the emergency error information of the node corresponding to the CiA Draft Standard 301. However, if the output CONFIRM is set to TRUE, the network layer's receiving buffer does not contain any emergency messages.

The function block always returns the first emergency message entered into the receiving buffer (= oldest message), the message is subsequently erased from the receiving buffer. Thus every emergency message can only be read one time by the PLC program. The function blocks CAN_RECV_EMCY_DEV and CAN_RECV_EMCY both access the same receiving buffer.

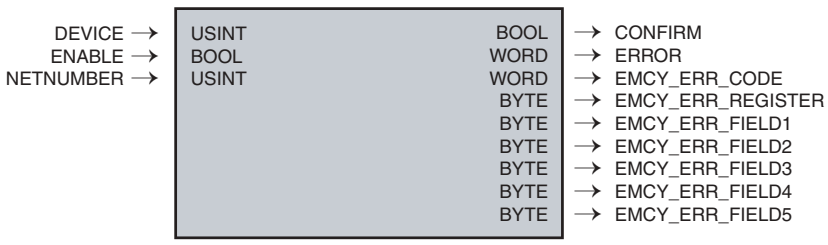
This function block is only available on control units in "PLC with CANopen Master" mode.

4-1-19 CAN_RECV_EMCY_DEV

Function block for reading emergency messages of a specific node from the receiving buffer of the network layer.

FB Prototype

CAN_RECV_EMCY_DEV



Input description

Label	Type	Description
DEVICE	USINT	Node address (1...127), for which the receipt of emergency messages is to be tested
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”
EMCY_ERR_CODE	WORD	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_REGISTER	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD1	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD2	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD3	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD4	BYTE	Emergency error information corresponding to the CiA Draft Standard 301
EMCY_ERR_FIELD5	BYTE	Emergency error information corresponding to the CiA Draft Standard 301

Description

The function block CAN_RECV_EMCY_DEV is used to read the emergency messages of a specific node from the network layer's receiving buffer. If upon return of the function block the output CONFIRM is set to TRUE, the elements EMCY_ERR maintain the emergency error information of the node corresponding to the CiA Draft Standard 301. If however the output CONFIRM is set to FALSE, then the receiving buffer of the network layer does not contain any emergency messages for the node in question.

The function block always returns the first emergency message entered into the receiving buffer (= oldest message), the message is subsequently erased from the receiving buffer. Thus every mergency message can only be read one time by the PLC program. The function blocks CAN_RECV_EMCY_DEV and CAN_RECV_EMCY both access the same receiving buffer. This function block is only available on one control unit in "PLC with CANopen Master" mode.

4-1-20 CAN_REGISTER_COBID

Function block for registering or erasing the receipt of PDOs and CAN Layer 2 messages via the network layer.

FB Prototype

CAN_REGISTER_COBID

Input description

Label	Type	Description
COBID	UINT	COBID of the message to be newly entered into registration or to be erased from the registration
REGISTER	BOOL	Enter COBID into/from registration
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
STATE	WORD	State or error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”

Description

The function block CAN_REGISTER_COBID is used for registration of a PDO or a CAN Layer 2 message for receipt via the network layer or for erasing such a registration. When the component is called with the REGISTER input set to TRUE, the COBID (CAN identifier) entered for receiving messages in the network layers is registered.

When the component is called and REGISTER = FALSE, the registration of the effected COBID is erased. If a component is called and REGISTER = FALSE and COBID = 0, all registrations are erased along with the messages stored in the network layer.

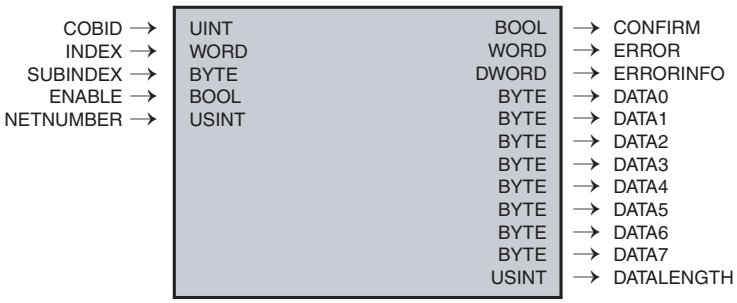
The network layer supports access to PDOs and CAN Layer 2 messages via a call of the function block CAN_PDO_READ8 generally only for registered messages.

4-1-21 CAN_SDO_READ8

Function block for reading a node’s object entries by way of an SDO transfer.

FB Prototype

CAN_SDO_READ8



Input description

Label	Type	Description
COBID	UINT	COBID of the message to be newly entered into registration or to be erased from the registration
INDEX	WORD	Number of the index entry to be read

SUBINDEX	BYTE	Number of the subindex entry to be read
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to data type “CIA405_CANOPEN_KERNEL_ERROR”
ERRORINFO	DWORD	Reserved for additional error information
DATA0	BYTE	Data byte of the received CAN message
DATA1	BYTE	Data byte of the received CAN message
DATA2	BYTE	Data byte of the received CAN message
DATA3	BYTE	Data byte of the received CAN message
DATA4	BYTE	Data byte of the received CAN message
DATA5	BYTE	Data byte of the received CAN message
DATA6	BYTE	Data byte of the received CAN message
DATA7	BYTE	Data byte of the received CAN message

Description

The function block CAN_SDO_READ8 is used to read the object entries of a node currently being used by the SDO transfer. The SDO transfer is always executed in the background for synchronization between the function block and the PLC program has to be applied while using both parameters ENABLE and CONFIRM. If the output CONFIRM is set to TRUE upon the return of the function block, the elements DATA0 through DATA7 receive the individual bytes of the object entry that was read. The output ATALENGTH reports the number of valid data bytes (beginning at DATA0).

The network layer supports only a single SDO transfer through the PLC program at any one time. After the start of the SDO transfer by setting ENABLE to TRUE, the SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer.

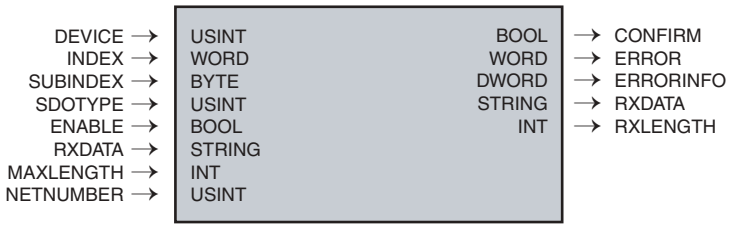
A call of the function block with DEVICE = 0 leads to an access of the local Object Dictionary of the PLC. Thus values from the local Object Dictionary can also be read.

4-1-22 CAN_SDO_READ_STR

FB for reading strings from the Object Dictionary of a node via SDO transfer.

FB Prototype

CAN_SDO_READ_STR



Input description

Label	Type	Description
DEVICE	USINT	Address of the node to be read
INDEX	WORD	Number of the index entry to be read
SUBINDEX	BYTE	Number of the subindex entry to be read

SDOTYPE	USINT	Type of the SDO transfer mode to be used corresponding to the data type "CAN_SDO_TYPE"
ENABLE	BOOL	Input for enabling or locking the function block
RXDATA	STRING	String variables for storing the read characters
MAXLENGTH	INT	Limit to the number of characters to be read, with 0 the buffer length of the transmitted string is calculated internally and used as a limt to the number of characters to be read

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”
ERRORINFO	DWORD	SDO abord code of the communication partner corresponding to the data type “CIA405_SDO_ERROR”
RXDATA	STRING	String variables for storing the read characters
RXLENGTH	INT	Length of the character sequence read

Description

The function block CAN_SDO_READ_STR is used to read the strings from a node's Object Dictionary utilizing SDO transfer. The SDO transfer is always executed in the background, thus the procedure described in section 4.1.3 for synchronization between the function block and the PLC program has to be applied while using both parameters ENABLE and CONFIRM. If the output CONFIRM is set to TRUE upon the return of the function block, the string given as element RXDATA contains the character sequence of the read object entry. The output RXLENGTH gives thereby the number of read characters (corresponds LEN(RXDATA);).

The network layer suppors only a single SDO transfer through the PLC program at any one time. After the start of the SDO transfer by setting ENABLE to TRUE, the SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer.

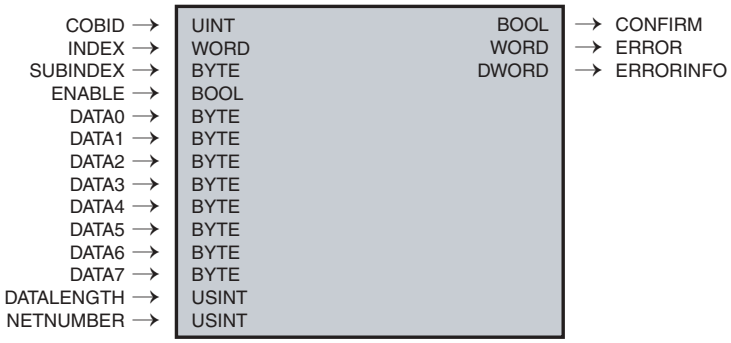
A function block call with DEVICE = 0 leads to an access of the local Object Dictionary of the PLC. Thus values from the local Object Dictionary can also be read.

4-1-23 CAN_SDO_WRITE8

Function block for writing object entries of a node by way of an SDO transfer.

FB Prototype

CAN_SDO_WRITE8



Input description

Label	Type	Description
COBID	UINT	COBID of the message to be newly entered into registration or to be erased from the registration
INDEX	WORD	Number of the index entry to be read
SUBINDEX	BYTE	Number of the subindex entry to be read

ENABLE	BOOL	Input for enabling or locking the function block
DATA0	BYTE	Data byte of the entry to be written
DATA1	BYTE	Data byte of the entry to be written
DATA2	BYTE	Data byte of the entry to be written
DATA3	BYTE	Data byte of the entry to be written
DATA4	BYTE	Data byte of the entry to be written
DATA5	BYTE	Data byte of the entry to be written
DATA6	BYTE	Data byte of the entry to be written
DATA7	BYTE	Data byte of the entry to be written
DATALENGTH	USINT	Length of the entry to be written
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to data type “CIA405_CANOPEN_KERNEL_ERROR”
ERRORINFO	DWORD	SDO abort code of the communication partner corresponding to the data type “CIA405_SDO_ERROR”

Description

The function block CAN_SDO_READ8 is used to read the object entries of a node currently being used by the SDO transfer. The SDO transfer is always executed in the background, thus the procedure described in section 4.1.3 for synchronization between the function block and the PLC program has to be applied while using both parameters ENABLE and CONFIRM. The individual bytes of the object entry to be written are transferred to the elements DATA0 through DATA7. Whereby the input DATALENGTH specifies the number of valid data bytes (beginning with DATA0).

The network layer only supports a single SDO transfer through the PLC program at anyone time. After the start of the SDO transfer by setting ENABLE to TRUE, this SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer.

A function block call with DEVICE = 0 leads to access of the local Object Dictionary of the PLC. Thus values can be written to the local Object Dictionary as well.

4-1-24 CAN_SDO_WRITE_STR

FB for writing strings to the Object Dictionary of a node via SDO transfer.

FB Prototype

CAN_SDO_WRITE_STR

Input description

Label	Type	Description
DEVICE	USINT	Address of the node to be read
INDEX	WORD	Number of the index entry to be written to
SUBINDEX	BYTE	Number of the subindex entry to be written to
SDOTYPE	USINT	Type of the SDO transfer mode to be used corresponding to the data type “CAN_SDO_TYPE”
ENABLE	BOOL	Input for enabling or locking the function block

TXDATA	STRING	String variables for storing the characters to be written
TXLENGTH	INT	Limit to the number of characters to be written, with 0 the length of the characters contained in the string TXDATA calculated internally (corresponds to LEN(TXDATA);)) and used as the number of characters to be writen
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”
ERRORINFO	DWORD	SDO abord code of the communication partner corresponding to the data type “CIA405_SDO_ERROR”
TXDATA	STRING	String variables for storing the characters to be written

Description

The function block CAN_SDO_WRITE-STR is used to write strings to the Object Dictionary of a node using SDO transfer. The SDO transfer is always executed in the background, thus the procedure described in section 4.1.3 for synchronization between the function block and the PLC program has to be applied while using both parameters ENABLE and CONFIRM.

The string to be written to the Object Dictionary must be given to the element TXDATA. The input TXLENGT specifies the number of valid characters. If this value is 0, then the length of the character sequence in the string TXDATA is determined internally (corresponds to LEN(TXDATA;) and is appled as the number of characters to be written. In this case the entire string contents are written.

The network layer suppors only a single SDO transfer through the PLC program at any one time. After the start of the SDO transfer by setting ENABLE to TRUE, the SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer.

The call of the function block with DEVICE = 0 leads to an access of the local Object Dictionary on the PLC. Thus values from the local Object Dictionary can be written as well.

4-1-25 CAN_SEND_SYNC

Function block for sending an individual SYNC message.

FB Prototype

CAN_SEND_SYNC

Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
NETNUMBER	USINT	Network number

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type “CIA405_CANOPEN_KERNEL_ERROR”

Description

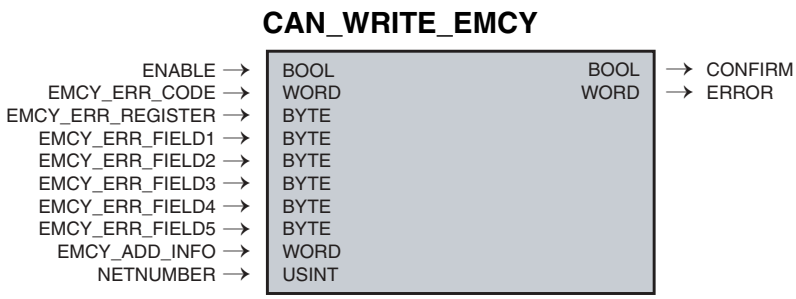
The function block CAN_SEND_SYNC is used to generate individual SYNC messages under complete control of the PLC program. With every component call where the input ENABLE is set to TRUE, a SYNC message is sent. Through this targeted influence it is possible to generate SYNC messages only if really relevant data has been altered.

This function block is only available on control units in “PLC with CANopen Master” mode and can only be used as alternative to the CAN_ENABLE_CYCLIC_SYNC function block.

4-1-26 CAN_WRITE_EMCY

Function block for sending application specific Emergency-Messages throught the network layer.

FB Prototype



Input description

Label	Type	Description
ENABLE	BOOL	Input for enabling or locking the function block
EMCY_ERR_CODE	WORD	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_REGISTER	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_FIELD1	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_FIELD2	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_FIELD3	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_FIELD4	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send
EMCY_ERR_FIELD5	BYTE	Emergency error information corresponding to the CiA Draft Standard 301 for the Emergency-Message to send

EMCY_ADD_INFO	WORD	Additional, application specific emergency error information, which is stored in index 1003H of the Object Dictionary (Error Field, see CiA Draft Standard 301), is not part of the Emergency-Message, thus it is only used for diagnostic purposes and can be set to zero

Output description

Label	Type	Description
CONFIRM	BOOL	Output for signal service completion by the function block
ERROR	WORD	Error code corresponding to the data type "CIA405_CANOPEN_KERNEL_ERROR"

Description

The function block CAN_WRITE_EMCY is used for sending application specific Emergency-Messages throught the network layer.

The EMCY_ERR members have to be filled with emergency error information according to the IEC61131-3 standard. The emergency message to send is generated from these members. The contents EMCY_ADD_INFO input can be defined by the application. This element is only stored in index 1003H of the Object Dictionary (Error Field, see CiA Draft Standard 301). It is not part of the Emergency- Message, thus it is only used for diagnostic purposes and can be set to zero. The index 1003H of the Object Dictionary can be read out by configuration and diagnostic tools.

When the function block CAN_WRITE_EMCY is called, the message to be sent gets stored in the send buffer of the CANopen kernel. If no error occurs here (message could be written to the buffer correctly), then the component returns with the output CONFIRM set to TRUE.

However, no return message is given to the PLC program showing whether or not the message could be sent successfully.

4-1-27 GET_SUPPORTED_DIAG

This FB can retrieve information about the type of diagnostic active on a given network node.

FB Prototype

GET_SUPPORTED_DIAG

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
NODE_ID	USINT	Network node	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
SUPPORTED_DIAG	USINT	Diagnostic Information Form	

Description

The node diagnostic data assigns individual bits with the following meaning:

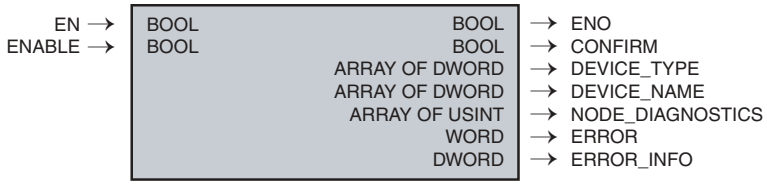
Description	Code
The form is available on the network	0x20
The module supports the Heartbeat	0x01
The module supports the Node Guarding	0x02

4-1-28 NET_ASCON_ID

This function block, starting from the scan performed by the master, provides information about the connected modules, about their diagnostic type, as they were recorded by the master and their identity. The function block is non-blocking and follows the calling rules of the CAN communication FBs.

FB Prototype

NET_ASCON_ID



Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
ENABLE	BOOL	Enabling the function of block	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/ FALSE
CONFIRM	BOOL	Result of the of module comparison	
DEVICE_TYPE	ARRAY OF DWORD	Content OD 1000h	
DEVICE_NAME	ARRAY OF DWORD	Content OD 1008h; max. 4 characters	
NODE_DIAGNOSTICS	ARRAY OF USINT	Producer and Diagnostics details	
ERROR	WORD	Error Code	
ERROR_INFO	DWORD	Details of the errors occurred	

Description

The data type arrays are composed of 127 elements from a number of possible addresses that correspond to the slave. The diagnostic data of node assigns individual bits mean the following:

Description	Code
The form is available on the network	0x20
The module on the network is Ascon	0x10
The module supports the Heartbeat	0x01
The module supports the Node Guarding	0x02

4-1-29 NG_RTR

Using the present Function Block it is possible to send an RTR to a network node respecting the Node Guardin protocol.

FB Prototype

NG_RTR

Input description

Label	Type	Description	Range
EN	BOOL	Enable In	TRUE/FALSE
NODE_ID	USINT	Network node	TRUE/FALSE

Output description

Label	Type	Description	Range
ENO	BOOL	Enable Out	TRUE/FALSE
OPRS	USINT	FB execution result	

Description

The node diagnostic data assigns individual bits with the following meaning:

Description	Code
Node address not valid	0x20
Node not controlled	0x01
Stack problem	0x02

Chapter 5
PROFIBUS (for CU-02 unit only)

The Ascon spa *sigmadue* series device **CU-02/2010** provides the *PROFIBUS-DP* connection capability as a slave unit. The communication protocol is Master/Slave and the CU-02 unit is able to provide functionalities for diagnostics, configuration and cyclic data exchange. The protocol itself does not take care about the meaning of the data exchanged, but it requires a configuration session where it is defined the exchanged data size. *Please check the specific Ascon spa documentation for details about the GSD file and configuration description.*

5-1 PROFIBUS-DP Slave Agent

For the PROFIBUS communication a %M memory area of 976 byte is reserved for the output data (from the CU-02 to the master PROFIBUS). These bytes are spitted in 4 different pages of 244 byte. For the input data (from the master PROFIBUS to the CU-02), the reserved memory size is 244 byte. The following table gives the memory map dedicated to the PROFIBUS interface.

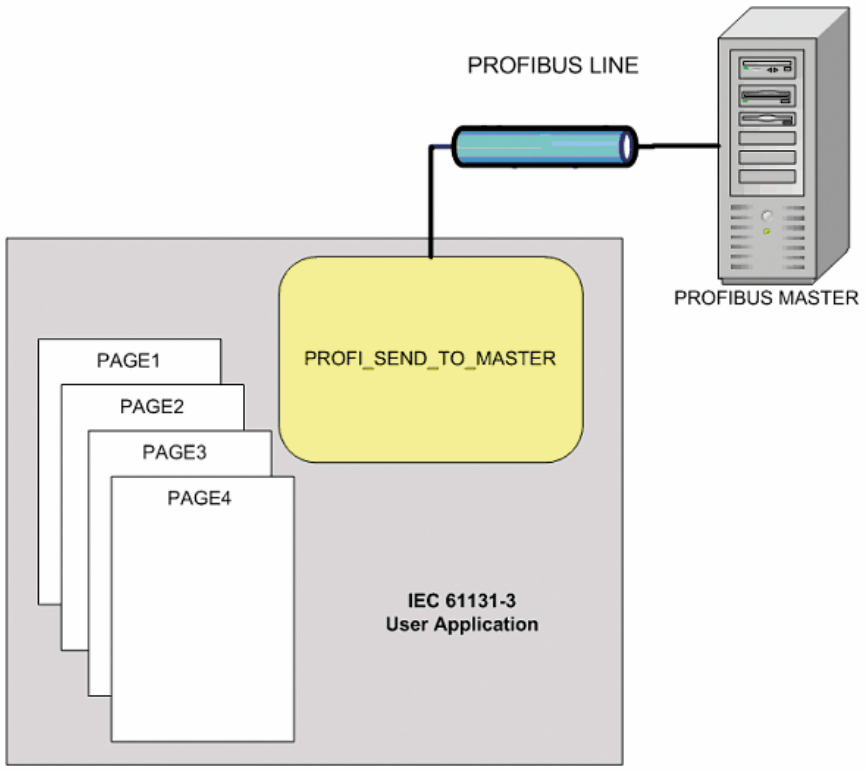
Profibus Agent	Protocol Memory Areas	PLC Memory Mapping
Profibus slave output	Page 1	From %M20000.0 to %M20243.0
	Page 2	From %M20244.0 to %M20487.0
	Page 3	From %M20488.0 to %M20731.0
	Page 4	From %M20732.0 to %M20975.0
Profibus slave input		From %M21000.0 to %M21243.0

The data exchange between master and slave could be managed in three different ways.

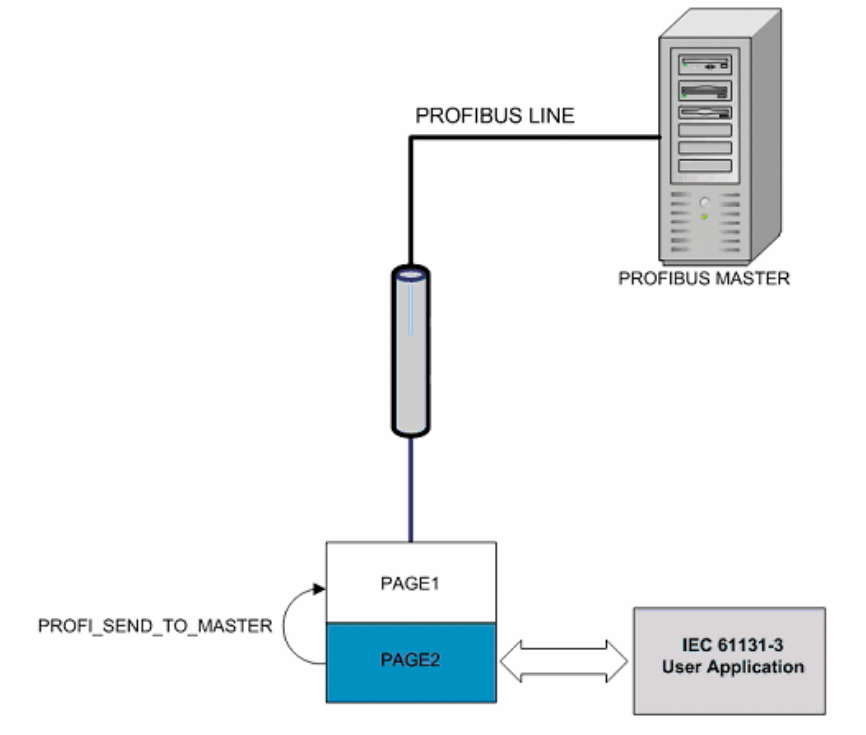
The first is the **basic mode**: it consists to enable the local PROFIBUS agent and to use only the page 1 to send data to the master (from %M20000.0 to %M20243.0). The user will fill this memory in order to send the correct data to the master.

The second is the **paged mode**: in this case the agent allows the user to fill all the four pages, and also to decide which page will be transferred in the next PROFIBUS telegram (using the PROFI_SEND_TO_MASTER). In this mode it is possible to send 976 bytes to the master.

Note: CU-02 can manage a total amount of 350 data bytes max. (considering MASTER_IN + MASTER_OUT).



The third is the **non paged mode with a data integrity control**: in this case the user updates always the data in the page 2 and using the function block PROFI_SEND_TO_MASTER (where in this case the parameter PAGE is meaningless) the CPU will copy all the data in the page 1 before to send the PROFIBUS telegram to the master. In this way the user control directly the data sent to the master, and it is not possible that a page is sent before the user does not finish to fill all the data of the page.



5-1-1 PROFI_ENABLE

FB Prototype

PROFI_ENABLE

Input Parameters

Label	Type	Description	Range
SLAVE_ID	USINT	Profibus Slave Node Address	0... 255
MASTER_IN	USINT	Master Input Byte Number	0... 244
MASTER_OUT	USINT	Master Output Byte Number	0... 244
PAGED	BOOL	Page Mode Enable	TRUE,FALSE

Output Parameters

Label	Type	Description	Range
OPRS	USINT	Execution Result	**0**: OK **1**: Init Error **2**: Size Error **3**: Hardware Error

Description

This function block allows the user to create an instance of the PROFIBUS agent on the CU-02 unit. The user must provide the node address (SLAVE_ID), the number of output bytes for the master (MASTER_IN - from the CU-02 to the master PROFIBUS), and the number of input bytes for the data coming from the master (MASTER_OUT - from the master PROFIBUS to CU-02). It is important to note that it is a user responsibility to take care about the total dimensions of the PROFIBUS telegrams, as specified in the PROFIBUS specification.

The user may specify also the mode for the data exchange with the master using the PAGED input, as reported in the following table:

Exchange mode	PAGED	Memory Space Used	Use of PROFI_SEND_TO_MASTER
Basic	FALSE	%M20000.0... %M20243.0	NO
Paged	TRUE	%M20000.0... %M21243.0	YES
Non paged mode with a data integrity control	FALSE	%M20244.0... %M20487.0	YES

Notes: **1.** To enable and start the Profibus communications, this Function Block **must be** called only one time.

2 CU-02 can manage a total amount of 350 data bytes max. (considering MASTER_IN + MASTER_OUT).

5-1-2 PROFI_STATUS

FB Prototype

PROFI_STATUS

Input Parameters

Label	Type	Description	Range
STATUS	USINT	Profibus slave agent status	**0**: Waiting for config data from the master **1**: PROFIBUS agent not instantiated **2**: Data Exchange running **3**: Protocol Error **4**: Hardware Error
PAGE_IN_USE	USINT	Page in use	1...4

Description

The PAGE_IN_USE parameter gives the indication of which output page actually is exchanged with the master (from CU-02 to the master). The following table gives the possible values for the different data exchange modes:

Exchange Mode	PAGE_IN_USE
Basic	1
Paged	1…4
Non paged mode with a data integrity control	2

5-1-3 PROFI_SEND_TO_MASTER

FB Prototype

PROFI_SEND_TO_MASTER

Input Parameters

Label	Type	Description	Range
PAGE	USINT	Page sent to the master PROFIBUS	1... 4

Output Parameters

Label	Type	Description	Range
OPRS	USINT	Execution Result	**0**: OK **1**: Page number Error **4**: Hardware Error

Description

The function block prepares the output data page sent to the master PROFIBUS unit. Depending on the Exchange Mode chosen, the PAGE value could assume different meanings:

Exchange Mode	PAGE value
Basic	FB not used
Paged	The value identifies the next exchanged page
Non paged mode with a data integrity control	The value it is not considered – Always the page2 will be copied over the page1

Chapter 6
Memory Copy Functions

In order to simplify the data exchange between the % variables it is available a set of dedicated function blocks that allow the users to move entire data blocks from one to another area.

6-1 Copy Function Blocks

6-1-1 MEMCOPY_FROM_M

FB Prototype

MEMCOPY_FROM_M

Input description

Label	Type	Description	Range
MEM_ADDR	UDINT	%M memory location to be copied [num]	0... 43000
DATATYPE	UDINT	Selection of the desired output data - type [num]	0 = TO_BYTE, 1 = TO_WORD, 2 = TO_DWORD

Output description

Label	Type	Description	Range
ERROR	BOOL	Error status	
ERR_CODE	DWORD	Error code [bit mask]	16#00 00 00 00… 16#FF FF FF FF
TO_BYTE	BYTE	Value copied from the desired memory location as 8 bit notation [num]	
TO_WORD	WORD	Value copied from the desired memory location as 16 bit notation [num]	
TO_DWORD	DWORD	Value copied from the desired memory location as 32 bit notation [num]	

Description

This function block performs a copy of a value from a valid desired memory location into a desired data type. The format selected as output data - types refers to the basic bit mask notation of the data. It needs then to be converted into the desired enumerated data - type (e.g. if 8 bit to USINT or if 16 bit to INT or if 32 bit to REAL, etc.).

Default Values

Input	Default Value
MEM_ADDR	22000
DATATYPE	0

Reference Table

Output	Description
ERR_CODE.0	Memory location out of range
ERR_CODE.1	Invalid datatype selection
...	...
ERR_CODE.31	

6-1-2 MEMCOPY_TO_M

FB Prototype

MEMCOPY_TO_M

Input description

Label	Type	Description	Range
MEM_ADDR	UDINT	%M memory location to be copied [num]	0 … 43000
DATATYPE	USINT	Selection of the desired output data type [num]	0 = TO_BYTE, 1 = TO_WORD, 2 = TO_DWORD
BYTE_VAL	BYTE	8 bit notation value to be copied at the desired memory location [num]	
WORD_VAL	WORD	16 bit notation value to be copied at the desired memory location [num]	
DWORD_VAL	DWORD	32 bit notation value to be copied at the desired memory location [num]	

Output description

Label	Type	Description	Range
ERROR	BOOL	Error status	
ERR_CODE	DWORD	Error code [bit mask]	16#00 00 00 00 … 16#FF FF FF FF

Description

This function block performs a copy of a specific data - type value at the desired valid memory location. The format selected as input data - types refers to the basic bit mask notation of the data. It needs then to be converted from the original enumerated data - type (e.g. if USINT into BYTE or if INT into WORD or if REAL into DWORD, etc).

Default Values

Input	Default Value
MEM_ADDR	22000
DATATYPE	0
BYTE_VAL	0
WORD_VAL	0
DWORD_VAL	0

Reference Table

Output	Description
ERR_CODE.0	Memory location out of range
ERR_CODE.1	Invalid datatype selection
ERR_CODE.2	Last writing memory location out of range
…	…
ERR_CODE.31	

6-1-3 MEMCPY_I_TO_M

FB Prototype

MEMCPY_I_TO_M

Input description

Label	Type	Description	Range
DEST_ADDR	UDINT	%M memory destination address [num]	All available addresses
SOURCE_ADDR	UDINT	%I (input) source address [num]	All admitted addresses
MEM_SIZE	UDINT	Amount of bytes to be copied. **The value, combined with the destination address (DEST_ADDR + MEM_SIZE), cannot exceed the last memory location available for the specific memory type** (I, M or Q) [num]	0... 43000

Output description

Label	Type	Description
ERROR	BYTE	Error status

Description

This function block performs a copy of a desired amount of bytes from a valid %I input memory area to a desired %M memory location.

Default Values

Input	Default Value
DEST_ADDR	0
SOURCE_ADDR	0
MEM_SIZE	0

Reference Table

Output	Description
OPRS	0 = OK, 1 = Source area outside the allowed address, 2 = Destination area outside the allowed address

6-1-4 MEMCPY_M_TO_M

FB Prototype

MEMCPY_M_TO_M

Input description

Label	Type	Description	Range
DEST_ADDR	UDINT	%M memory destination address [num]	All available addresses
SOURCE_ADDR	UDINT	%M (input) source address [num]	All admitted addresses
MEM_SIZE	UDINT	Amount of bytes to be copied. **The value, combined with the destination address (DEST_ADDR + MEM_SIZE), cannot exceed the last memory location available for the specific memory type** (I, M or Q) [num]	0... 43000

Output description

Label	Type	Description
ERROR	BYTE	Error status

Description

This function block performs a copy of a desired amount of bytes from a valid %M memory area to a desired %M memory location.

Default Values

Input	Default Value
DEST_ADDR	0
SOURCE_ADDR	0
MEM_SIZE	0

Reference Table

Output	Description
OPRS	0 = OK, 1 = Source area outside the allowed address, 2 = Destination area outside the allowed address

6-1-5 MEMCPY_M_TO_Q

FB Prototype

MEMCPY_M_TO_Q

Input description

Label	Type	Description	Range
DEST_ADDR	UDINT	%Q memory destination address [num]	All available addresses
SOURCE_ADDR	UDINT	%M (input) source address [num]	All admitted addresses
MEM_SIZE	UDINT	Amount of bytes to be copied. **The value, combined with the destination address (DEST_ADDR + MEM_SIZE), cannot exceed the last memory location available for the specific memory type** (I, M or Q) [num]	0... 43000

Output description

Label	Type	Description
ERROR	BYTE	Error status

Description

This function block performs a copy of a desired amount of bytes from a valid %M memory area to a desired %Q memory location.

Default Values

Input	Default Value
DEST_ADDR	0
SOURCE_ADDR	0
MEM_SIZE	0

Reference Table

Output	Description
OPRS	0 = OK, 1 = Source area outside the allowed address, 2 = Destination area outside the allowed address

6-1-6 MEMCPY_Q_TO_M

FB Prototype

MEMCPY_Q_TO_M

Input description

Label	Type	Description	Range
DEST_ADDR	UDINT	%M memory destination address [num]	All available addresses
SOURCE_ADDR	UDINT	%Q (input) source address [num]	All admitted addresses
MEM_SIZE	UDINT	Amount of bytes to be copied. **The value, combined with the destination address (DEST_ADDR + MEM_SIZE), cannot exceed the last memory location available for the specific memory type** (I, M or Q) [num]	0... 43000

Output description

Label	Type	Description
ERROR	BYTE	Error status

Description

This function block performs a copy of a desired amount of bytes from a valid %Q memory area to a desired %M memory location.

Default Values

Input	Default Value
DEST_ADDR	0
SOURCE_ADDR	0
MEM_SIZE	0

Reference Table

Output	Description
OPRS	0 = OK, 1 = Source area outside the allowed address, 2 = Destination area outside the allowed address

Chapter 7
Real Time Clock

The Ascon CPUs are equipped with a clock powered by the unit battery which is thus always active. Two function blocks (RTC_SETUP and RTC_GET_VALUES) are used to configure and read the clock. The clock sets the time and date as follows:

- Seconds
- Minutes
- Hours
- Day of month
- Day of week
- Month
- Year.

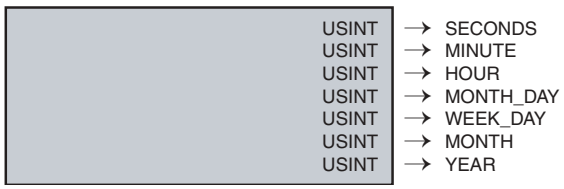
The clock only stores the last two digits of the year. Hence, the year 2008 is expressed as the number 8.

7-1 Description of the individual Function Blocks

7-1-1 RTC_GET_VALUES

FB Prototype

RTC_GET_VALUES



Output description

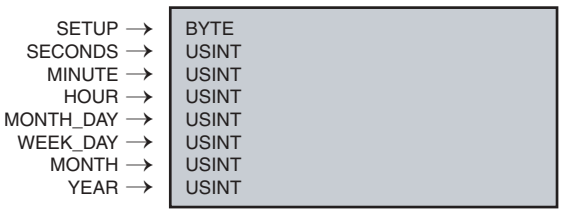
Label	Type	Description	Range
Seconds	USINT	Actual seconds read	0... 59
Minute	USINT	Actual minutes read	0... 59
Hour	USINT	Actual hours read	0... 23
Month_day	USINT	Actual day of month read	1... 31
Week_day	USINT	Actual day of week read	1... 7
Month	USINT	Actual month read	1... 12
Year	USINT	Actual mear read	00... 99

Description This function block can be used to verify the Real Time Clock date and time values available within the **sigmadue** CPU models.

7-1-2 RTC_SETUP

FB Prototype

RTC_SETUP



Input description

Label	Type	Description	Range
Setup	BYTE	Mask defining the data to be set [bit]	16#00... 16#7F
Seconds	USINT	Seconds to be set [num]	0... 59
Minute	USINT	Minutes to be set [num]	0... 59
Hour	USINT	Hours to be set [num]	0... 23
Month_day	USINT	Day of month to be set [num]	1... 31
Week_day	USINT	Day of week to be set [num]	1... 7
Month	USINT	Month to be set [num]	1... 12
Year	USINT	Year to be set, evaluated as 2000 + YEAR, e.g. 11 means 2011 [num]	00... 255

Description

This function block can be used to setup the internal Real Time Clock available within the **sigmadue** CPU models. It is not necessary to assign all the values together but it can be done parameter by parameter accordingly the bit mask specified with the SETUP function block input (see the above “Reference table”). If desired, it is possible to set all the parameters together by setting the SETUP value to 16#7F (hex) or 2#1111111 (bin).
It is not necessary to set just one parameter at a time; they can all be assigned in a single call by setting the setup mask to 0xF or 0x7.

Default Values

Input	Default Value
SETUP	0
SECONDS	0
MINUTE	0
HOUR	0
MONTH_DAY	0
WEEK_DAY	0
MONTH	0
YEAR	0

Reference Table

SETUP bit	Value (hex or bin)	Description
0	16#01 or 2#1	SECONDS
1	16#02 or 2#10	MINUTE
2	16#04 or 2#100	HOUR
3	16#08 or 2#1000	MONTH_DAY
4	16#10 or 2#10000	WEEK_DAY
5	16#20 or 2#100000	MONTH
6	16#40 or 2#1000000	YEAR
7	16#80 or 2#10000000	N.A.

Chapter 8
Watchdog

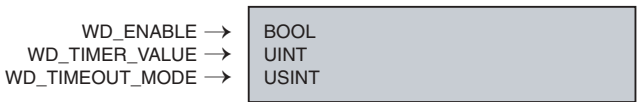
The Ascon CPU units are equipped with a watchdog mechanism controlled by two function blocks. The watchdog is a decreasing counter which is decremented every 100ms. When the counter reaches zero, the system could respond in two different modes: store the event and open a dedicated digital output, or perform a system reset. Please note that even the watchdog mechanism is controlled by the function blocks, its operation is independent from the PLC running program. This means that during the debug of an application a PLC stop command does not stop the watchdog counter, which will terminate according to the programmed mode.

8-1 Description of the individual Function Blocks

8-1-1 WATCHDOG_SET

FB Prototype

WATCHDOG_SET



Input description

Label	Type	Description	Range
WD_ENABLE	BOOL	Enable/Disable the Watchdog operations	True/False
WD_TIMER_VALUE	UINT	Sets the timer countdown value [100 ms]	0... 65535
WD_TIMEOUT_MODE	USINT	Sets the watchdog operation mode [num]	0... 1

Description

This function block can be used to setup the watchdog control available within the **sigmadue** CPU models. When the function is activated by using it, the designed digital output, if enabled, is forced to 1 (closed). The specified countdown time value is calculated and reset at every cycle of the application before it reaches 0 (zero) (for this reason the FB should not be used in a timer task and the timer value should be evaluated accordingly the maximum cycle time of the application to avoid side effects…). If something happens and the timer cannot be reset (basically it reaches 0) the digital output will be deactivated by forcing it to 0 (opened).

Default Values

Input	Default Value
WD_ENABLE	FALSE

Input	Default Value
WD_TIMER_VALUE	0
WD_TIMEOUT_MODE	0

Reference Table

WD_TIMEOUT_MODE	System operations when WD timer expires
0	System reset (same as power cycle)
1 (SigmaPAC CU-02)	The CPU opens the onboard Alarm output and the watchdog timeout event is stored
1 (μPAC MP-01)	The CPU opens the DO1 output (if enabled) and the watchdog timeout event is stored (please refer to the “MP-01 User Manual” on how to enable the DO1 as watchdog output).

8-1-2 WATCHDOG_STATUS

FB Prototype

WATCHDOG_STATUS

Output description

Label	Type	Description	Range
WD_TIMER	UINT	Actual timer value	
WD_EXPIRED	BOOL	Timer status	

Description

This function block can be used to monitor the watchdog status available within the **sigmadue** CPU models. When the function block is used, and the watchdog function is not used to control the specific digital output, it return a status at TRUE if something happens and the timer cannot be reset (basically it reaches 0). The status then can be used to perform a desired action.

Chapter 9
Random number generators

9-1 Description of the individual Function Blocks

9-1-1 RAND

Function Block needed to have random numbers from 0 to 65535.

FB Prototype

RAND

Output description

Label	Type	Description	Range
NUMBER	UINT	Random number	0..65535

Description

Sometimes having random numbers in the 1131-3 program can be useful. This Function Block generates random numbers ranging from 0 to 65535.

Chapter 10
Type conversion (FLATTEN IEEE – REAL and vice versa)

The following FBs can turn useful to transfer the value of a floating-point variable via modbus protocol. Particularly, it is possible to convert a REAL variable to its single byte equivalent in a DWORD variable and vice versa, before sending it via modbus line. The byte equivalent of a variable is indicated as FLATTEN.

10-1 Description of the individual Function Blocks

10-1-1 ASCON_FLATTEN_TO_REAL

FB Prototype

ASCON_FLATTEN_TO_REAL

Input description

Label	Type	Description	Range
INDWORD	DWORD	32 bit DWORD number representation to be converted [num]	0... 4294967295

Output description

Label	Type	Description	Range
OUTREAL	REAL	Converted REAL value [num]	$-3.4E^{-38}$... $3.4E^{+38}$

Description

This function block performs a conversion from a DWORD variable into a REAL data type value compliant to the IEEE-754 specification, which provides the following representation regarding the 32 bit floating point numbers:

Bit	31	30...23	22... 0
Meaning	Sign: 0 = positive, 1 = negative	Exponent calculated as value of this field - 127	Mantissa

10-1-2 ASCON_REAL_TO_FLATTEN

FB Prototype

ASCON_REAL_TO_FLATTEN

Input description

Label	Type	Description	Range
INREAL	REAL	Number to be converted into a REAL value [num]	$-3.4E^{-38}$... $3.4E^{+38}$

Output description

Label	Type	Description	Range
OUTWORD	DWORD	Converted 32 bit DWORD number representation [num]	0... 4294967295)

Description

This function block performs a conversion from a REAL data type value compliant to the IEEE-754 specification, which provides the following representation regarding the 32 bit floating point numbers, into the corresponding 32 bit DWORD number representation:

Bit	31	30...23	22... 0
Meaning	Sign: 0 = positive, 1 = negative	Exponent calculated as value of this field - 127	Mantissa

Chapter 11
Counter Block (for MP-01 unit only)

On the MP-01 unit it is possible to combine a counter for each local digital input: in the configuration session the counter function is enabled (please check the "**MP-01User Manual**" for details) and in the application code the user can read the values. A reset function block is provided in order to reset the counter value in the application code.

11-1 RESET_PULSE_COUNTER

The function block reset the counter value of the specified digital input.

FB Prototype

RESET_PULSE_COUNTER

Input description

Label	Type	Description	Range
CH_IN	USINT	Channel number	1...8

Output description

Label	Type	Description	Range
OPRS	BYTE	Execution Result	**0**: OK **2**: Wrong Channel Number indication

Description

The function block reset the counter value of the specified digital input. In case of error it returns an error code.

Chapter 12
TC_Calibrate (for MP-01 unit only)

Function Block to recalculate the compensation caused by the sensor break detection method.

FB Prototype

TC_CALIBRATE

Input description

Label	Type	Description	Range
EXEC_CALIB	BOOL	Command flag	TRUE, FALSE
CHANNEL	USINT	Channel number	1... 2

Output description

Label	Type	Description	Range
OPRS	BYTE	**0**: OK **1:** TC input channel not present **2**: Wrong Channel Number indication	0... 2

Appendix A
Reference documents

[1] "*Infoteam OpenPCS programming system – user manual*" – version 6.0 english

[2] "*IEC 61131-3: Programming Industrial Automation Systems*" – Karl-Heinz John, Michael Tiegelkamp - Springer

[3] "*Ascon Firmware Function Block Library*"

[4] "*IEC 61131-3 Function Block Library*".

[5] "*Estensioni per gestire porte di comunicazione dell'ambiente OpenPCS*" V1.0 – Maurizio Grassi

[6] *CANopen Extension for IEC61131 – Software manual*" – Edition March 2005 – Systec Electronic

[7] "*CiA Draft Standard 405 – CANopen Interface and Device Profile for IEC61131-3 Programmable Devices*" – version 2.0

[8] "*CANopen Application Layer and Communication Profile - CiA DS301 v 4.02*"

[9] "*Modbus Messaging on TCP/IP implementation guide*" - http://www.Modbus-IDA.org

[10] "*MODBUS over Serial Line Specification & Implementation guide*" - http://www.Modbus-IDA.org

[11] "*MODBUS APPLICATION PROTOCOL SPECIFICATION*" - http://www.Modbus-IDA.org

[12] "*CU-02 Installation manual*" (code: J30 - 658 - 1ACU02 E).

[13] "*CU-02 User manual*" (code: J30 - 478 - 1ACU02 E).

[14] "*MP-01 Installation manual*" (code: J30 - 658 - 1AMP01 E).

[15] "*MP-01 User manual*" (code: J30 - 478 - 1AMP01 E).

[16] "*micro**PAC** I/O modules Installation Manuals*".

[17] "*micro**PAC** I/O modules User Manuals*".